VIRTUALIZATION

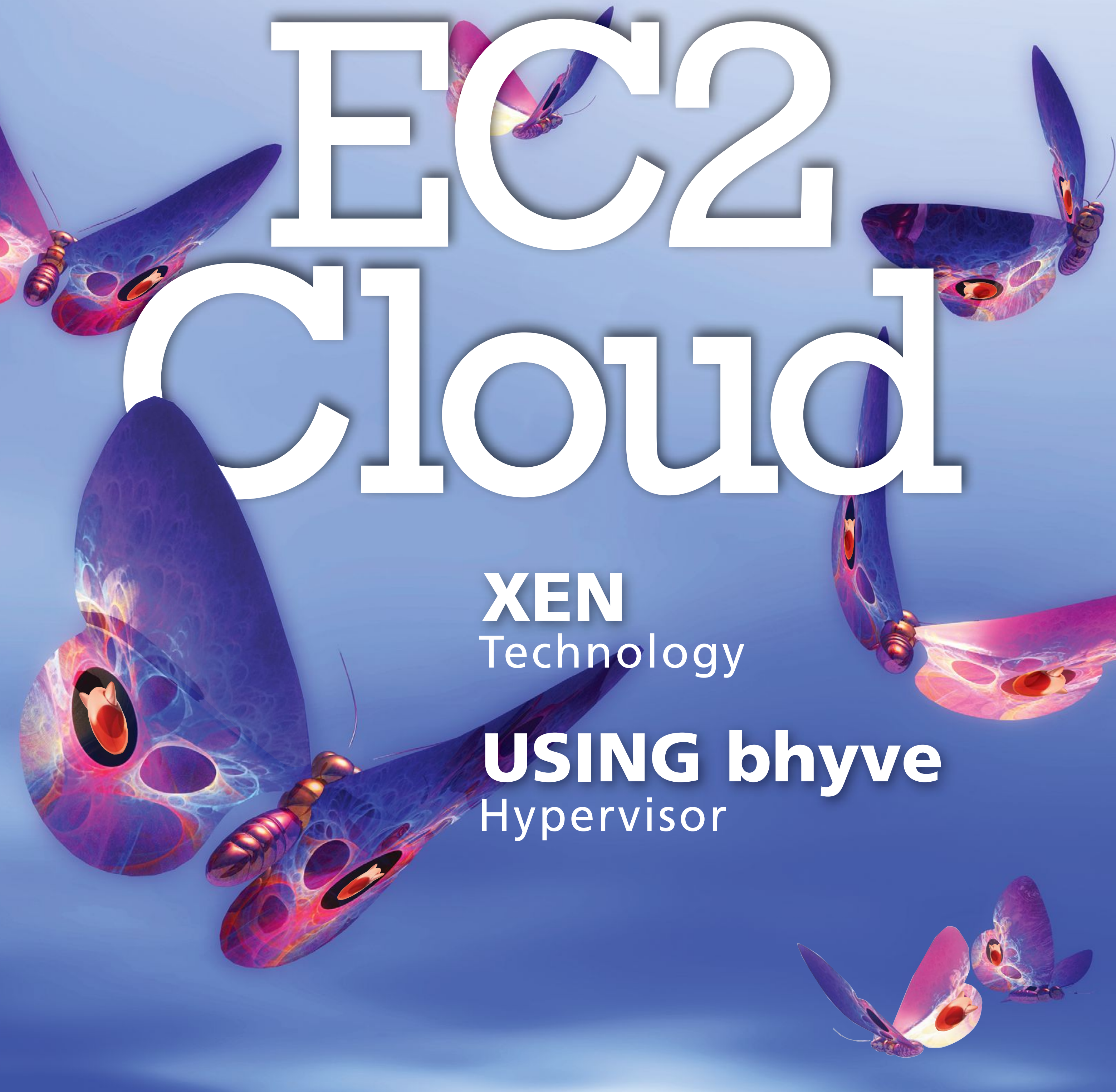
FreeBSD™ JOURNAL

July/August 2014
Vol. 1 • Issue No. 4

FreeBSD in the Amazon EC2 Cloud

XEN Technology

USING bhyve Hypervisor



(pfSense)10

We're taking pfSense®
to a higher power.

Secure your network with pfSense® 2.2 and FreeBSD® 10 on the new Netgate® C2758.

With 8 Intel® processing cores, up to 8 GbE ports, AES-NI and multi-threaded packet filtering, we're advancing network security to a higher power.

Netgate®

7212 McNeil Drive Suite 204 Austin, TX 78729 +1.512.646.4100

Table of Contents

FreeBSD Journal
Vol. 1, Issue No. 4

FreeNAS
in an Enterprise Environment

By the time you're reading this, FreeNAS has been downloaded more than 5.5 million times. For home users, it's become an indispensable part of their daily lives, akin to the DVR. Meanwhile, all over the world, thousands of businesses universities, and government departments use FreeNAS to build effective storage solutions in myriad applications.

What you will learn...

- How TrueNAS builds off the strong points of the FreeBSD and FreeNAS operating systems
- How TrueNAS meets modern storage challenges for enterpri

The FreeNAS operating systems is freel
the public and offers thorough docu
active community, and a feature-rich
the storage environment. Based on FreeBS
can share over a host of protocols (SMB,
FTP, iSCSI, etc) and features an intuitive
the ZFS file system, a plug-in system fo
much more.
Despite the massive popularity of
aren't aware of its big brother dutifu
data in some of the most demandin
environments: the proven, enterpri
professionally-supported line of a
But what makes TrueNAS differe
Well, I'm glad you asked...

Commercial Grade Suppo

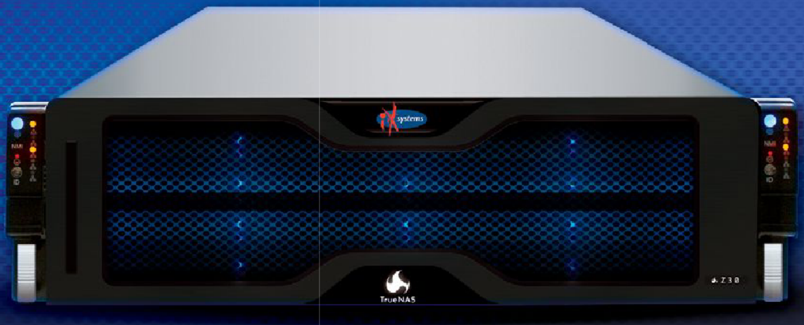
When a mission critical storag
organization's whole operatio
halt. Whole community-base
free), it can't always get an
and running in a timely ma
responsiveness and expert
dedicated support team c
provide that safety.
Created by the same
developed FreeNAS.

NEW RELEASE

iX systems®

WE INTERRUPT THIS MAGAZINE TO BRING YOU THIS IMPORTANT ANNOUNCEMENT:

THE PEOPLE WHO DEVELOP FREENAS, THE WORLD'S MOST POPULAR STORAGE OS, HAVE JUST REVAMPED TRUENAS.



**POWER WITHOUT CONTROL MEANS NOTHING.
TRUENAS STORAGE GIVES YOU BOTH.**

- ✓ Simple Management
- ✓ Hybrid Flash Acceleration
- ✓ Intelligent Compresssion
- ✓ All Features Provided Up Front (no hidden licensing fees)
- ✓ Self-Healing Filesystem
- ✓ High Availability
- ✓ Qualified for VMware and HyperV
- ✓ Works Great With Citrix XenServer®

To learn more, visit: www.iXsystems.com/truenas

intel® Xeon® inside™

POWERED BY INTEL® XEON® PROCESSORS

S&W PUBLISHING LLC
PO BOX 408, BELFAST, MAINE 04915

Publisher •	Walter Andrzejewski walter@freebsdjournal.com
Editor-at-Large •	James Maurer jmaurer@freebsdjournal.com
Art Director •	Dianne M. Kischitz dianne@freebsdjournal.com
Office Administrator •	Cindy DeBeck cindy@freebsdjournal.com
Advertising Sales •	Walter Andrzejewski walter@freebsdjournal.com Call 888/290-9469

FreeBSD Journal (ISBN: 978-0-615-88479-0) is published 6 times a year (January/February, March/April, May/June, July/August, September/October, November/December).

Published by the FreeBSD Foundation, PO Box 20247, Boulder, CO 80308
ph: 720-207-5142 • fax: 720-222-2350
e-mail: board@freebsdfoundation.org
Copyright © 2014 by FreeBSD Foundation. All rights reserved.

This magazine may not be reproduced in whole or in part without written permission from the publisher.

We're past the halfway point of our first year and *FreeBSD Journal*™ is doing well.

Subscriptions have grown quickly, and for that we would like to thank the readers and authors who have joined in our effort to produce a world-class magazine by and about FreeBSD.

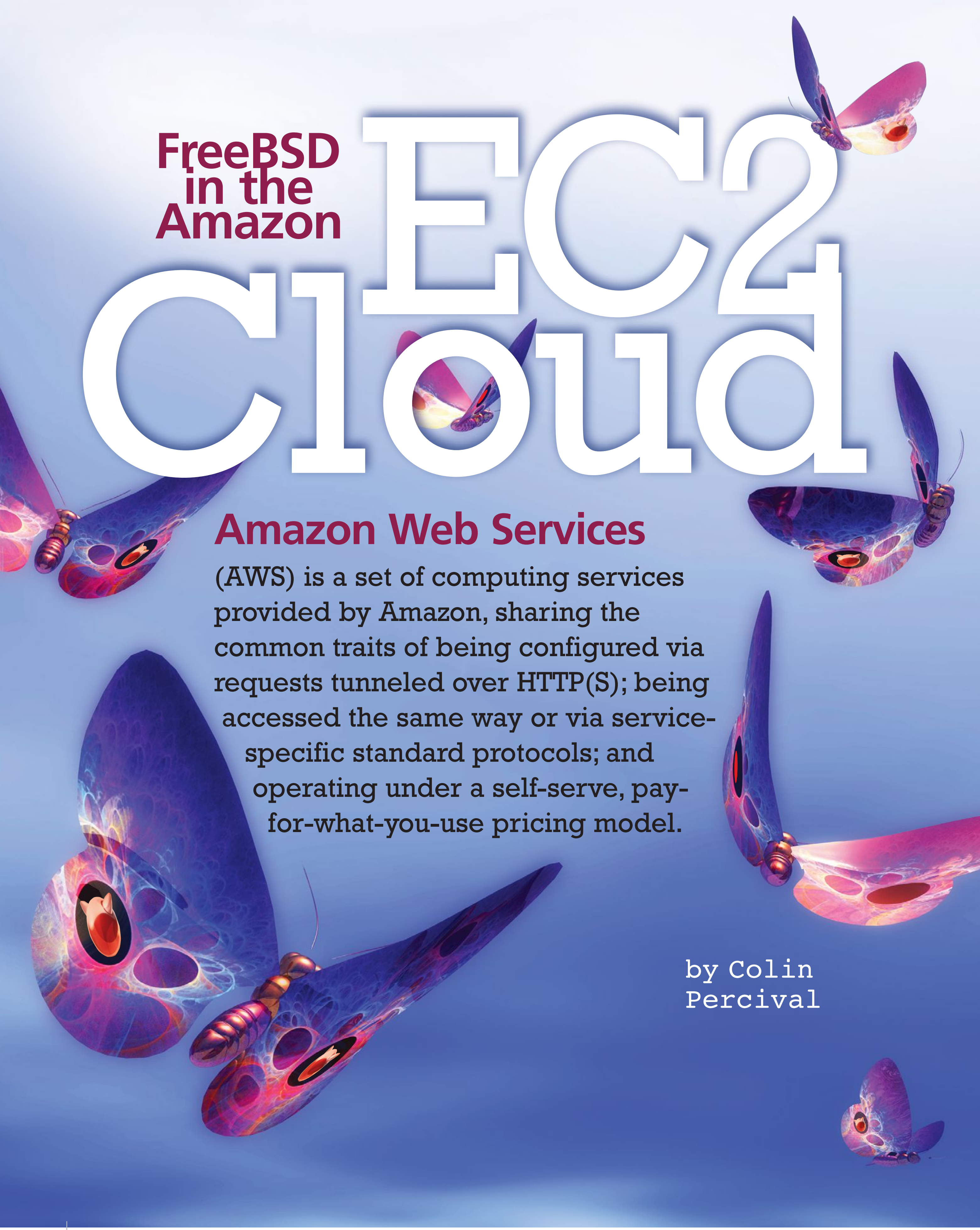
Virtualization has been a hot topic ever since servers ran out of megahertz but continued to sprout cores. For this, our fourth issue, we've put together a slate of articles that show FreeBSD in virtualized environments and FreeBSD as the virtualization technology. Colin Percival takes us through FreeBSD on EC2, Amazon's cloud service, and Roger Pau Monné discusses FreeBSD and Xen. FreeBSD also has its own virtualization technology, bhyve, originally developed at NetApp, and then open-sourced and made available directly within FreeBSD. John Baldwin explains how to set up and work with your own virtual instances of FreeBSD within bhyve.

Outside the virtualization theme, Brendan Gregg writes about the USE Method for applying various tools to finding performance issues. USE stands for "Utilization, Saturation, and Errors."

We bid farewell and offer our thanks to Thomas Abthorpe, who wrote the first three Ports Report columns. We welcome our newest columnist, Frederic Culot, who is taking over that same column. And don't miss our first Conference Report—this one on BSDCan and contributed by Michael Dexter.

We're happy to consider new columns and your article ideas. If you have an inclination to write, email a title and abstract, or even just a paragraph explaining your idea to editor@freeBSDJournal.com. Now we're all off to work on Issue #5.

Sincerely,
FreeBSD Journal Editorial Board



FreeBSD in the Amazon EC2 Cloud

Amazon Web Services (AWS) is a set of computing services provided by Amazon, sharing the common traits of being configured via requests tunneled over HTTP(S); being accessed the same way or via service-specific standard protocols; and operating under a self-serve, pay-for-what-you-use pricing model.

by Colin Percival

Drawing on Amazon's experience building and operating computing infrastructure for their retail operations, AWS was the first entrant into the modern cloud computing industry and remains the largest player. Amazon provides dozens of services, but the most widely used ones can be placed into four major categories:

Core infrastructure services

- Elastic Compute Cloud (EC2)–Virtual machines
- Simple Storage Service (S3)–Durable data storage with ~50 ms response time
- Glacier–Low-cost durable data storage with ~4 hour response time
- DynamoDB–High-performance durable NoSQL data store
- Simple Queue Service (SQS)–Reliable message queueing [1]

Edge location services

- Route 53–Authoritative DNS
- CloudFront–HTTP(S) content distribution network

Managed software services

- ElastiCache–Memcached or Redis
- Relational Database Service (RDS)–MySQL, PostgreSQL, Oracle, or SQL Server
- Elastic Map Reduce (EMR)–Hadoop

AWS usability services

- Identity and Access Management–Create subaccounts with specified privileges
- CloudFormation–Scripted deployment of a set of AWS resources
- CloudWatch–Gathering and monitoring of AWS and custom metrics
- CloudTrail–AWS API call logging

All of these services are accessed via API calls made using a pair of AWS "access keys." An "Access Key ID," which is sent as part of the requests to identify the account, and a "Secret Access Key," which is used to create a cryptographic signature that authenticates the API request [2]. Amazon provides libraries for several programming languages to facilitate use of these APIs in addition to a command line utility and a web interface that can be used to manage many services. This article is concerned only with the Elastic Compute Cloud service, but readers interested in other AWS services are encouraged to consult Amazon's excellent documentation [3].

[1] While message queuing may seem like esoteric functionality, having reliable queues can be a great boon to designers of distributed systems where nodes might die and leave work half-done if it is not retained in a reliable queue.

[2] In addition to the fully privileged "root" access keys for an account, the AWS Identity and Access Management service can be used to create restricted "users" with access keys able to issue only a specified set of requests.

[3] http://aws.amazon.com/documentation/

[4] In keeping with Amazon's low-overhead, self-serve model, credit cards are the only payment method accepted except for US government users and companies spending over $5,000/month.

[5] For production use, you would probably want to use the Identity and Access Management to create a "User" which has a restricted set of privileges, and then create a key pair for that subaccount. For simplicity, we'll use the "root" account keys in this article.

Getting Started with AWS

Before you can use EC2 (or any other AWS services) you will need to create an AWS account. Point a web browser at http://aws.amazon.com/ and click on the "Sign Up" button. You can then log in with an existing amazon.com account or createa new one.

Once you've created or logged into an amazon.com account, you will be asked to provide your name and contact details, accept the AWS Customer Agreement, provide a credit card for billing [4], and complete an "identity verification" process consisting of receiving a phone call and entering a four digit PIN to activate your account for Amazon Web Services.

Now that you have an AWS account, you will need keys for accessing it [5]. Point a web browser at https://console.aws.amazon.com/iam/home?#security_credential and select "Access Keys" and "Create New Access Key." Make a note of the Access Key ID and the Secret Access Key and create a file named `~/.aws/config` containing the following lines (you will need to create the `~/.aws/` directory first):

```
[default]
aws_access_key_id = <Your Access Key ID goes here>
aws_secret_access_key = <Your Secret Access Key goes here>
region = us-east-1
```

Finally, you need software for accessing the AWS services. In this article we're going to use the AWS Command Line Interface (http://aws.amazon.com/cli/). To install this on your FreeBSD system run:

`# make -C /usr/ports/devel/awscli install clean` or
`# pkg install awscli` as root.

Amazon Elastic Compute Cloud (EC2)

The *Amazon Elastic Compute Cloud* (EC2) is, at its core, rent-by-the-hour virtual machines running under the Xen hypervisor. When EC2 was first launched in 2006, the idea of renting "virtual servers" was already well established, as many companies offered such services using either FreeBSD jails or virtualization systems such as Xen, but such offerings typically involved paying for a month or more at a time, and the role of virtual servers was limited to being a cheaper alternative to dedicated servers. By introducing hourly billing for virtual machines and providing an API for quickly and automatically provisioning new virtual machines, EC2 made available an entirely different benefit of virtualization–flexibility. Rather than renting servers for months at a time [6], EC2 is designed to allow users to scale up and down rapidly in response to load–hence the "Elastic" in "Elastic Compute Cloud."

Launching an EC2 *instance*–or indeed any sort of virtual machine–requires specifying three important parameters: Where to launch the virtual machine, what type of virtual machine to launch, and what should run on the virtual machine. Amazon Web Services, including EC2, is split into 8 Regions: North Virginia, Oregon, North California, Ireland, Singapore, Tokyo, Sydney, and São Paulo [7]. With the exception of some "support" services like billing and authentication systems, each region functions independently of the others. Consequently, in addition to the obvious benefit of reducing latency by placing servers close to systems with which they will be communicating, EC2 regions can also provide benefits for redundancy and regulatory compliance (e.g., ensuring that EU data never leaves the EU). Regions have names like "us-east-1".

Within each AWS Region, there are two or more EC2 *Availability Zones*. These exist for the purpose of making redundant systems possible: Amazon does its best to ensure that any failures (power, networking, etc.) can only affect a single availability zone. Availability zones have names like "us-east-1a".

When EC2 first launched, it only offered a single type of virtual machine, but over the years their offerings have expanded to 38 types, with names like *m3.medium*, *c3.xlarge*, or *i2.8xlarge* (in general, <family>.<size>). While there is not space in this article to provide details about all of them, there is an important technical detail concerning the virtualization technology—FreeBSD

[6] For customers who can commit to long-term use, Amazon also offers "reserved instances," which are akin to buying hardware and then paying a heavily discounted rate for power and networking. These can offer substantial savings over the standard "on-demand" hourly rates for users with consistent usage patterns.

[7] For more details about AWS regions, see http://aws.amazon.com/about-aws/globalinfrastructure/regional-product- services/ .

[8] When EC2 launched, it only supported Xen "PV" mode. Amazon added limited support for HVM in order to support Windows, but did not retrofit their existing systems to support it.

[9] Users who will be launching a large number of identical instances (e.g., spinning up web and application servers based on fluctuating traffic levels) often create customized AMIs with code and configuration preinstalled. In such cases there will not necessarily be any need to log in via SSH, of course.

Name	# CPUs	RAM	Solid state disks	Price (us-east-1 region)
t2.micro	10%	1.0 GB	none	$0.013/hour
t2.small	20%	2.0 GB	none	$0.026/hour
t2.medium	2 x 20%	4.0 GB	none	$0.052/hour
m3.medium	1	3.75 GB	4 GB	$0.070/hour
m3.large	2	7.5 GB	32 GB	$0.140/hour
m3.xlarge	4	15.0 GB	80 GB	$0.280/hour
m3.2xlarge	8	30.0 GB	160 GB	$0.560/hour
c3.large	2	3.75 GB	32 GB	$0.105/hour
c3.xlarge	4	7.5 GB	80 GB	$0.210/hour
c3.2xlarge	8	15.0 GB	160 GB	$0.420/hour
c3.4xlarge	16	30.0 GB	320 GB	$0.840/hour
c3.8xlarge	32	60.0 GB	640 GB	$1.680/hour
r3.large	2	15.0 GB	32 GB	$0.175/hour
r3.xlarge	4	30.5 GB	80 GB	$0.350/hour
r3.2xlarge	8	61.0 GB	160 GB	$0.700/hour
r3.4xlarge	16	122.0 GB	320 GB	$1.400/hour
r3.8xlarge	32	244.0 GB	640 GB	$2.800/hour
i2.xlarge	4	30.5 GB	800 GB	$0.853/hour
i2.2xlarge	8	61.0 GB	1600 GB	$1.705/hour
i2.4xlarge	16	122.0 GB	3200 GB	$3.410/hour
i2.8xlarge	32	244.0 GB	6400 GB	$6.820/hour

requires support for Xen "HVM" mode, which the older EC2 instance types only provide in combination with a Windows license [8]. Most people will want to use one of the 21 instance types belonging to the "current generation" M3 ("general purpose"), C3 ("high CPU"), R3 ("high RAM"), I2 ("high I/O"), or T2 ("low cost") families–which provide better price/performance levels than the older instances anyway. See chart on page 6.

Note that with the exception of the amount of SSD storage on the "medium" and "large" sizes, each step size within a family doubles the number of CPUs, amount of RAM, amount of SSD storage, and hourly price. Conversely, comparing the xlarge sizes of different families, we see that the "high CPU" family is identical to the "standard" family except with half the RAM and a 25% discount on price. The "high RAM" moves in the opposite direction, with double the RAM of "standard" instances and a price 25% higher; and "high I/O" instances are "high RAM" instances with a tenfold increase in SSD storage–and a further 145% increase in price.

The T2 family is different from other EC2 instance types, in that it has "burstable" CPUs rather than dedicated CPUs: Over the long run they can only use a CPU for 10% or 20% of the time, but if a CPU is underused a "balance" will accumulate for up to a day. These instances can be very useful for systems which are mostly idle but occasionally need to rebuild ports or other similarly CPU-taxing work. (See http://aws.amazon.com/blogs/aws/low-cost-burstable-ec2-instances/ for details).

Once you have decided where to launch an instance and what type of instance to launch, you'll need to tell EC2 what software the instance should run. In EC2 jargon, this is an ***Amazon Machine Image*** (AMI) and it consists of a disk image (containing FreeBSD, for example) and some metadata which tells EC2 how to launch the image. AMIs can be created directly from disk snapshots in EC2 or by "re-bundling" a running EC2 instance–but new users will find it easiest to start by using one of the thousands of existing, freely available AMIs.

Because EC2 operates in a "cloud" environment, there are two more considerations required to securely connect to and use a new virtual machine–firewall rules and login credentials [9]. All EC2 instances are launched into "security groups," which is a fancy way of saying that a firewall rule set can be created in advance and applied to a new instance. By default, new security groups allow all outgoing traffic, but block all incoming traffic except responses to outgoing connections [10]. Finally, EC2 provides a mechanism for specifying an SSH public key for logging in to an instance. This key is provided (along with a variety of other metadata) over a special

[10] The firewall is stateful, but not perfectly so. In particular, in some cases it will block incoming ICMP fragmentation required packets, with the effect of breaking Path MTU Discovery. You may wish to add a firewall rule to allow incoming ICMP type 3 code 4 packets.

[11] The opposite applies if you're launching a large number of identical instances between which you will have traffic load-balanced. In that case you should put them all into the same Security Group so that you can adjust the firewalls for all of them at once.

Launching a FBSD EC2 Instance

In this example we will be launching an m3.medium instance costing $0.07/hour. If you forget to shut down the instance, it will continue running and your credit card will be billed roughly $50/month—so don't forget!

If you followed the instructions under "Getting started with AWS," you should now have the awscli port installed and a configuration file in `~/.aws/config`. Before we launch an EC2 instance, we need to create an SSH key we will use to log in to it. Run the following commands:

```
$ ssh-keygen -f ~/.ssh/ec2login
$ aws ec2 import-key-pair --key-name mykey --public-key-material file://`realpath
~/.ssh/ec2login.pub`
```

This will create an SSH key (in the file "ec2login") and send the public part up to EC2, associating it with the name "mykey." (The `file://` mess is because the `awscli` code expects URIs for everything rather than simple paths to files. I suggested changing this, but the author felt that using URIs provided better consistency.)

We're also going to create an EC2 security group. For historical reasons there is a pre-existing "default" group, but it's a good idea to create a new security group when you're launching an instance for a new purpose. That way you can adjust its firewall rules later without affecting other instances [11]. Run the following commands to create a security group and allow SSH connections from your IP address to its instances:

```
$ aws ec2 create-security-group --group-name mygroup --description “my security group”
$ aws ec2 authorize-security-group-ingress --group-name mygroup --protocol tcp --port
22 --cidr <your IP address>/32
```

Now to launch an instance: Normally at this point you would want to look up which AMI you want to launch–there is a list at http://www.daemonology.net/freebsd-on-ec2/ with AMI IDs for different versions of FreeBSD in each AWS region, but for this example we'll use FreeBSD 10.0-RELEASE in the us-east-1 region (which is the default value we selected when we created the ~/.aws/config file)—that AMI is ami-69dae900.

Launching Instance continues next page.

Launching continued

Run the following command to launch FreeBSD 10.0-RELEASE into an m3.medium virtual machine which we will use the mykey SSH key to connect into, and output the instance ID. [12]

```
$ aws ec2 run-instances --image-id ami-69dae900 --instance-type m3.medium --key-
name mykey —security-groups mygroup --output text --query 'Instances[*].InstanceId'
```

Make a note of the instance ID as you'll need it for the rest of the commands here. If you forget it, you can list all your running instance (and get lots of information about them) by running

```
$ aws ec2 describe-instances.
```

Now go make a cup of coffee. We need to kill about five minutes while FreeBSD boots and EC2 collects its console output for us [13].

You're back and caffeinated? Good, now you can run

```
$ aws ec2 get-console-output --output text --instance-id <Your Instance ID goes
here> | more
```

and you will see FreeBSD's usual boot-time kernel output. Scroll down, though, and you'll see a few things you don't get in a standard FreeBSD install. First, `freebsd-update` runs, downloading and installing critical updates to FreeBSD—important, since with a virtual machine in the "cloud" we can't exactly follow the usual security advice to log in and apply security updates before exposing a new server to the internet! After that, you'll see the `pkg` bootstrap running and the `awscli` package being downloaded and installed. We will see later how to change the set of packages installed on first boot. Finally, we see FreeBSD rebooting. This is done if `freebsd-update` installed any updates or any `rc.d` scripts were added by installed packages, so that we will have a system running up-to-date code with all enabled services.

It's interesting to see what FreeBSD is doing, but there's really only one vital reason to read the console output: To see the fingerprints of the SSH host keys. These host keys are generated when the system first boots, and–again, because this is in the cloud–we need to use this out-of-band mechanism to ensure that when we connect to the system we're SSHing into the right box. In addition to being printed when the keys are generated, the fingerprints are printed again with a prefix of `ec2:` in order to make them easily extracted by automatic tools [14].

Speaking of SSH, it's time to use it. Run:

```
$ aws ec2 describe-instances --output text --query
'Reservations[*].Instances[*].PublicIpAddress' --instance-id <Your Instance ID
goes here>
```

to get the public IP address of your instance. Then run:

```
$ ssh -i ~/.ssh/ec2login ec2-user@<Instance IP address goes here>
```

SSH should prompt you to confirm the SSH host fingerprint. You can compare this against the value you saw a moment ago in the console output.

You should now be logged in to your EC2 instance as the default unprivileged "ec2-user" user. To become root, simply run 'su' (there is no root password set in the standard image). This system will now behave like any other FreeBSD 10.0-RELEASE system. When you've finished looking around, log out.

Now unless you want to pad Amazon's profits, you should destroy your EC2 instance. Run the following commands:

```
$ aws ec2 modify-instance-attribute --block-device-mappings '[ { "DeviceName":
"/dev/sda1", "Ebs": { "DeleteOnTermination": true } } ]' --instance-id <Your
Instance ID goes here>
$ aws ec2 terminate-instances --instance-ids <Your Instance ID goes here>
```

The first command is necessary because the default behavior is to retain a copy of the boot disk of a terminated instance (costing $0.50/month for the standard 10-GB instance boot disk).

If you want clean up your AWS account completely, you can also delete the security group you created using the '`aws ec2 delete-security-group`' command and delete the SSH key using the '`aws ec2 delete-key-pair`' command, but these are not essential–Amazon doesn't charge anything for security groups or SSH keys.

management interface. In FreeBSD, it is code in the `sysutils/ec2-scripts` port which reads this data and arranges for user logins.

Elastic Block Store, Elastic IP Addresses, Elastic Load Balancer (oh my!)

As a platform for creating virtual machines, EC2 is useful enough, but there are several other services that add important functionality to EC2. First, ***Elastic Block Store*** (EBS), which makes it possible to create and attach virtual disks to EC2 instances, second, ***Elastic IP Addresses*** (EIP), which make it possible to reserve an IP address and assign it to an instance or move it between instances, and third, ***Elastic Load Balancers*** (ELB), which provide an easy mechanism for load balancing traffic between a pool of instances.

Elastic Block Store would probably have been better named EC2 Storage Area Network. With an API call you can create a virtual disk of between 1 GB and 1 TB, known as an ***EBS Volume***, within a specified EC2 availability zone. Another API call can be used to attach a volume to an EC2 instance. Because these volumes are accessed over Amazon's network rather than being physically connected to the boxes hosting EC2 instances, it is easy to detach a volume from one EC2 instance and attach it to another. This can help for migrating data to a new instance, or–as often happens to FreeBSD developers–if a system is accidentally rendered unbootable [15].

Elastic Block Store comes in three flavors: "Magnetic" (formerly known as "Standard"), "Provisioned IOPS," and "General Purpose." Magnetic volumes are cheap, but as the name implies they have typical "spinning iron oxide" performance levels–typically 50 to 100 I/O operations per second on 128 kB blocks. Provisioned IOPS volumes, in contrast, allow you to specify a target I/O rate of between 1 and 4,000 I/O operations per second on 16 kB blocks, but are more expensive, especially for large reserved I/O rates. General Purpose volumes fall into a middle ground: They can burst up to 3,000 I/O operations per second, but provide a performance baseline of 3 I/O operations per second per GB of allocated space. It is important to remember however that because EBS is accessed over the network, all flavors are slower than the SSD storage which is attached directly to EC2 instances–there is an unavoidable trade-off between optimizing for performance and optimizing for usability.

Elastic IP Addresses should have been named Reserved IP Addresses, since that's exactly what they are. Like EBS volumes, Elastic IP Addresses can be allocated, attached to EC2 instances, and moved between EC2 instances. These are necessary for directly public-facing servers, since without those EC2 instances are created with randomly selected IP addresses and there is no way for a new instance to assume the address of an earlier instance which has been shut down.

While Elastic IP Addresses make it possible to put an address into DNS and know that it can be pointed at a new EC2 instance if required, this does not satisfy the needs of companies with large numbers of public-facing servers. Enter Elastic Load Balancers: An Elastic Load Balancer is configured with a pool of EC2 instances–as usual, with API calls to add or remove EC2 instances–and it forwards incoming connections to instances from the pool.

There are many more EC2 features: Virtual Private Cloud, which allows you to configure virtual networks including routing tables, IP subnets, and VPN endpoints; Elastic Network Interfaces, which make it possible to attach multiple addresses to an EC2 instance; CloudWatch, which provides monitoring of EC2 instances; Autoscaling, which makes it possible to automatically launch or shut down instances in response to load. There is not enough space here to write in detail about everything, so we can merely refer the reader to Amazon's excellent website and documentation.

Instance Autoconfiguation Using configinit

While many people may be satisfied with launching a clean FreeBSD system and SSHing in to perform necessary system configuration, especially for one-off servers, there are advantages both in speed and repeatability to having some or all of the configuration process scripted and performed automatically when an EC2 instance first boots. Many Linux systems, including Ubuntu and RedHat, make use of the CloudInit system, which allows a user-data file provided when an EC2 instance is launched to specify a set of commands to be run.

CloudInit is not very well suited for use in FreeBSD for two reasons. First, it uses Python, which is not part of the FreeBSD base system [16], and second, because CloudInit is built around the concept of configuring a system by running commands. In contrast to Linux systems, FreeBSD is far more "configuration file

[12] The default behavior of the AWS CLI is to output JSON containing a very large number of parameters about the request completed. This is useful for programs which speak JSON, but for shell scripts I always use the `--output text` option along with `--query` to specify the particular details I want.

[13] It seems to take a minimum of 3 minutes before console output will be available via the EC2 API, and if you try to read the output too soon, EC2 seems to cache the "no output" response and you need to wait even longer.

[14] One such tool is the author's ec2-known-host script, which populates .ssh/known_hosts with EC2 host fingerprints, available at http://www.daemonology.net/blog/2012-01-16-automatically-populating-ssh-known-hosts.html.

[15] The process in this case is non-trivial, but it is possible to detach the boot disk from an instance, attach it temporarily to another instance so that something can be fixed, and then move it back and reboot the formerly non-booting instance.

[16] It isn't feasible to include Python in standard FreeBSD EC2 images either, since there are several different versions of Python in the FreeBSD ports tree, and the one installed to run CloudInit would inevitably not be the one people would want to use for other purposes.

oriented," and so I decided to write my own system, which I called "configinit." Unlike CloudInit, the configinit system takes the form of a very simple shell script, and is included in FreeBSD EC2 images starting from 10.0-RELEASE. Similar to CloudInit, however, configinit runs early in the boot process when an EC2 instance first launches, downloads the EC2 user data (which is exposed via the same management interface used for ssh public keys), and then processes that file.

There are four types of files that configinit can handle:

• If a file starts with the characters `>/` then the first line minus the leading `>` character will be interpreted as a path, and the rest of the file (everything except the first line) will be written to that location (replacing any existing file).

• If a file starts with the characters `>>/` then the first line minus the leading `>>` characters will be interpreted as a path, and the rest of the file will be appended to that location (creating a new file if no file yet exists at that location).

• If a file starts with the characters `#!` then the file will be executed (this is most likely to be used with shell scripts).

Otherwise, configinit attempts to extract the file as an archive using bsdtar (which automatically detects a wide range of archive and compression formats), and then recurses onto each file in lexicographical order.

The most straightforward use of this functionality is to add content to FreeBSD's "master configuration file," `/etc/rc.conf`. The configinit code instructs the `rc` system to reload that file in order to ensure that settings changed will be reflected later in the boot process. In addition to the standard FreeBSD configuration options (e.g., `sshd_enable="YES"`, which appears in the `rc.conf` file in EC2 images), there are a few useful options for the packages which are preinstalled on EC2 images:

```
firstboot_pkgs_list="list of packages"
```

will cause those packages to be downloaded

Setting Up Drupal at Instance Launch Using configinit

Now we're going to use configinit to launch a system with Drupal installed. The same caveat applies here as before–EC2 instances cost money, so make sure you don't forget to clean up after you've finished exploring

First, let's create a new SSH key pair for the instance we're going to launch. Although we're not going to SSH into the instance right now, if you were planning on using this you would need to be able to SSH in later to perform upgrades (of both FreeBSD and the packages required by Drupal) and to perform backups. Run the following commands:

```
$ ssh-keygen -f ~/.ssh/drupal
$ aws ec2 import-key-pair --key-name drupal --public-key-material file://`realpath ~/.ssh/drupal.pub`
```

We also need to create a new EC2 security group (aka. firewall rule set). In this case we want to open up port 80 to the entire world, while there's no need for SSH access. If and when you need to SSH into this instance later, you can use `authorize-security-group-ingress` to add a rule allowing incoming traffic on port tcp/22. Run the following commands:

```
$ aws ec2 create-security-group --group-name drupal --description "drupal demo security group"
$ aws ec2 authorize-security-group-ingress --group-name drupal --protocol tcp --port 80 --cidr
0.0.0.0/0
```

Now for the configinit data, rather than retyping the data run the following command to fetch a Drupal configinit file:

```
$ fetch http://freebsd-ec2-dist.s3.amazonaws.com/drupal-conf.tar
```

If you look inside that archive you'll see three files named "`rcconf`," "`apache`," and "`drupalinit`." The names aren't important; when configinit extracts the archive and processes the files individually, it will handle them in lexicographical order, but in this case the order doesn't matter. The file "`rcconf`" contains the following lines:

```
>>/etc/rc.conf
ec2_fetchkey_user="drupal-user"
firstboot_pkgs_list="apache22  drupal7  mod_php5  mysql55-server"
apache22_enable="YES"
mysql_enable="YES"
```

The first line means "append the rest of this file to `/etc/rc.conf`," while the remaining lines are rc.conf settings that tell the EC2 boot scripts to set up SSH logins for a user named "drupal-user." Download and install the packages `apache22`, `drupal7`, `mod_php5`, and `mysql55-server`, start `Apache 2.2` and start MySQL.

The file "`apache`" writes to `/usr/local/etc/apache22/Includes/local.conf` the necessary configuration to tell Apache to serve files out of `/usr/local/www/drupal7` (which is where the Drupal port installs its data), and to enable PHP (which Drupal uses).

The file "`drupalinit`" is a shell script which performs steps needed to allow Drupal to work. It sets ownerships to and installed when the EC2 instance first boots, `firstboot_freebsd_enable="NO"` willl disable the launch-time bootstrapping of the pkg system (useful if an instance is being launched into a networking environment where it cannot access the pkg mirrors), `firstboot_freebsd_update_enable="NO"` will disable the launch-time downloading and installation of critical errata and security updates, `ec2_fetchkey_user="username"` will change the name of the user created and configured for SSH access via the public key provided to EC2.

For example, providing the following user-data file

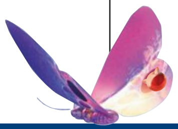
```
>>/etc/rc.conf
firstboot_pkgs_list="apache22 python33"
ec2_fetchkey_user="webmaster"
apache22_enable="YES"
```

will result in Apache 2.2 and Python 3.3 being preinstalled, a user named "webmaster" being created for SSH logins and Apache 2.2 being launched automatically during the boot process.

More sophisticated uses of configinit will generally need to create or edit multiple files. In those cases it will be necessary to create an archive file (e.g., a tarball) containing one file for each required configuration change (see "Setting up Drupal at Instance Launch" for an example).

Future Directions for FreeBSD/EC2

FreeBSD has been available on EC2 in some form since December 2010, and it has been stable enough for production use since mid-2011. A lot of progress has been made since then, and especially in the past year. In October 2012, Amazon announced the M3 instance family, followed in November by the C3 instance family, in December 2013 by the I2 instance family, in April 2014 by the R3 instance family, and in July 2014 the T2 instance family–and between them, this "current generation" of EC2



the "`www`" user so that the Drupal code (running via Apache and mod_php5) can function and creates a mysql database and user for Drupal to use. There's a catch, however. Because configinit runs early in the boot process–by necessity, early enough to add `rc.conf` settings which specify which packages to install–there is no mysql daemon running and the Drupalfiles which need to have ownership changed do not exist yet. To escape this limitation, the `drupalinit` script creates a new rc.d script, which will run after packages are installed and mysql is running–a script which then deletes itself after initializing Drupal data to prevent itself from being run again the next time the system boots.

Now that we've seen how the Drupal configinit file works, let's see it in action. Run the following command:

```
$ aws ec2 run-instances --image-id ami-69dae900 --instance-type m3.medium --key-name
drupal --security-groups drupal --user-data file://drupal-conf.tar --output text --
query 'Instances[*].InstanceId'
```

Note that the only change between this and launching a standard FreeBSD image is the added `--user-data file://drupal-conf.tar` option. As before, the clumsy `file://` syntax is required due to the URI-centric design of the AWS CLI.

As before, we'll need to wait for FreeBSD to boot and EC2 to collect its console output. After five minutes, run:

```
$ aws ec2 get-console-output --output text --instance-id <Your Instance ID goes here>
| more
```

and you'll see FreeBSD booting, updating itself, downloading and installing the packages we requested, rebooting, launching the Apache and MySQL daemons, and finally you'll see a line

```
Drupal password: <12 characters>
```

which is the (randomly generated) password for the MySQL "drupal" user. Once we have the password, we can proceed with configuring Drupal. Run:

```
$ aws ec2 describe-instances --output text --query 'Reservations[*].Instances[*].
PublicIpAddress' --instance-id <Your Instance ID goes here>
```

to get the public IP address of your instance, and then paste it into your web browser of choice. When prompted for database parameters, tell Drupal to use the database named "drupal," database username "drupal," and the password you obtained from the console output. Congratulations, you now have Drupal running and ready for you to add content to your new website!

As always with EC2 instances, when you've finished you'll want to clean up to avoid paying unnecessary costs. Run the following commands:

```
$ aws ec2 modify-instance-attribute --block-device-mappings '[ { "DeviceName":
"/dev/sda1", "Ebs": { "DeleteOnTermination": true } } ]' --instance-id <Your Instance
ID goes here>
$ aws ec2 terminate-instances --instance-ids <Your Instance ID goes here>
```

instance types make it possible to run FreeBSD across a wide range of hardware profiles without any of the evil hacks needed to run FreeBSD on earlier instance types.

Coming from the other direction, starting with FreeBSD 10.0-RELEASE in January 2014, FreeBSD gained Xen support in the GENERIC kernel configuration, allowing official release binaries to run in EC2 and making it possible to use the FreeBSD Update system for binary updates. The addition of configinit made it possible to configure FreeBSD virtual machines via EC2 user data and scripts are now available that allow anyone to build FreeBSD AMIs [17].

So what's next? This will depend largely on what FreeBSD–and EC2–users ask for, but some possible highlights include:

• Integrating the EC2 AMI building process into the FreeBSD release process and having AMIs built by the FreeBSD release engineering team,

• Providing EC2 AMIs with ZFS or MFS root filesystems,

• Automatically using EC2 ephemeral disks to cache or mirror data on EC2 Elastic Block Store volumes to provide greater performance and/or reliability [18],

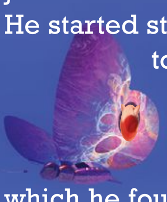
• Using Identity and Access Management "roles" to allow an EC2 instance to automatically create and add more Elastic Block Store volumes to itself, allowing for truly "elastic" file-systems, and

• Providing more configinit samples for ready-to-use server setups.

But more than anything else what FreeBSD/EC2 needs right now is more users, more testing, and more feedback. What works? Is there anything that doesn't work? What features would you like to see added? Most of the development is done. Now it's time for the baton to be passed to the wider user community–go forth and use! •

[17] http://www.daemonology.net/blog/2014-02-16-FreeBSD-EC2-build.html

[18] FreeBSD/EC2 already autoconfigures swap space on EC2 ephemeral disks, but most instance types have far more ephemeral disk space than swap space alone can use.



Colin Percival has been a FreeBSD developer since 2004 and was the project's Security Officer from 2005 to 2012. He started struggling to bring FreeBSD to the EC2 environment in 2006, and now uses it extensively in the Tarsnap online backup service, which he founded and continues to run.

RootBSD

Premier VPS Hosting

RootBSD has multiple datacenter locations, and offers friendly, knowledgeable support staff. Starting at just $20/mo you are granted access to the latest FreeBSD, full Root Access, and Private Cloud options.

www.rootbsd.net

FREE AND OPEN SOURCE EXPO AND TECHNOLOGY CONFERENCE

FOSSETCON 2014

Come out and participate in the First Annual Fossetcon 2014 Florida's Only Free and Open Source Conference. With in 10 minutes of Disney Land, Universal Studios and Epcot Center.

DAY 0 FOOD, TRAINING, WORKSHOPS AND CERTIFICATIONS

DAY 1 FOOD, KEYNOTES, EXPO HALL, SPEAKER TRACKS

DAY 2 FOOD, KEYNOTES, EXPO HALL, SPEAKER TRACKS

BSD *Friendly*

FREE FOOD, TRAINING, CERTIFICATIONS AND GIVEAWAYS!!!

SEP 11 - SEPT 13

ROSEN PLAZA HOTEL ORLANDO, FL

More info at **www.fossetcon.org**

Fossetcon 2014: The Gateway To The Open Source Community

XEN

BY ROGER PAU MONNÉ

THE XEN HYPERVISOR started at the University of Cambridge Computer Laboratory in the late 1990s under the project name Xenoservers. At that time, Xenoservers aimed to provide "a new distributed computing paradigm, termed 'global public computing,' which allows any user to run any code anywhere. Such platforms price computing resources, and ultimately charge users for resources consumed".

Given this goal, it is clear, Xen was designed for the cloud even before the cloud was created. Today Xen technology powers the biggest enterprise clouds in production, including Amazon EC2, RackSpace, and Verizon Terramark.

Using a hypervisor allows sharing the hardware resources of a physical machine among several OSes in a secure way. The hypervisor is the piece of software that manages all those OSes (usually called guests), and provides separation and isolation between them.

First released in 2003 as an open-source hypervisor under the GPLv2, Xen's design is OS agnostic, which makes it easy to add Xen support into new OSes. Since its first release more than 10 years ago, Xen receives broad support from a large community of individual developers and corporate contributors.

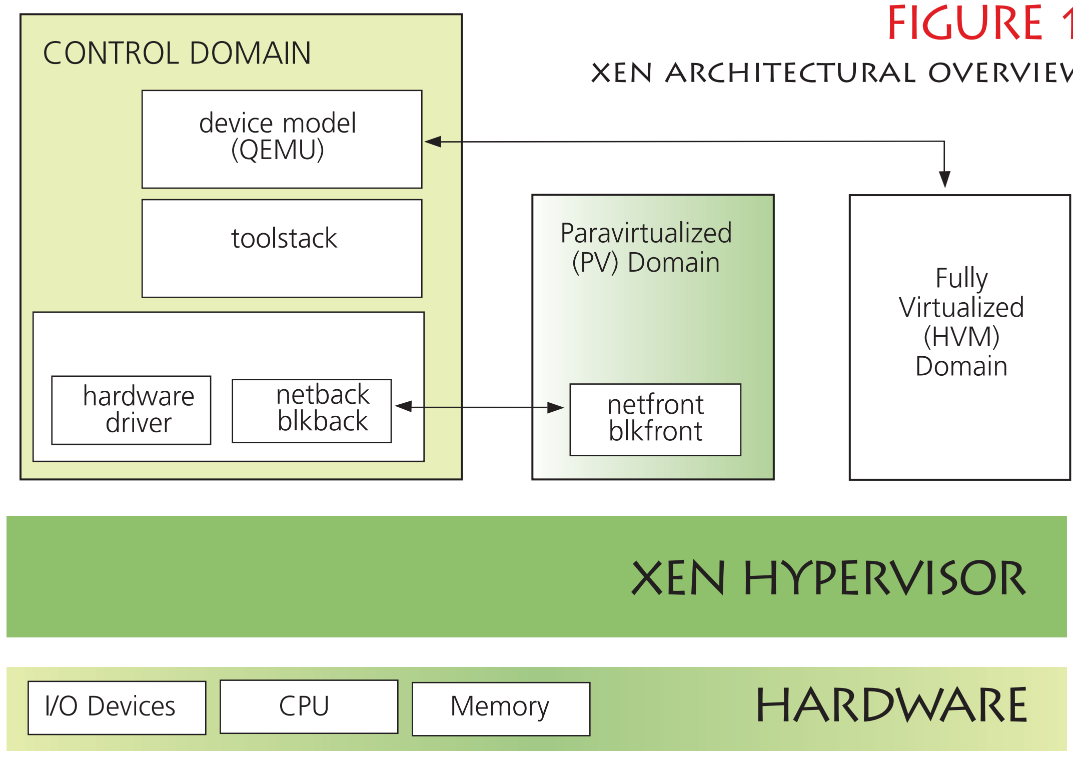
The Architecture

Hypervisors can be divided into two categories: Type1—those that run directly on bare metal and are in direct control of the hardware, and Type2—hypervisors that are part of an operating system. Common Type1 hypervisors are VMware ESX/ESXi and Microsoft Hyper-V, while VMware Workstation and VirtualBox are clear examples of Type2 hypervisors.

Xen is a Type1 hypervisor with a twist—its design resembles a microkernel in many ways. Xen itself only takes control of the CPUs, the local and IO APICs, the MMU, the IOMMU and a timer (either HPET or PIT). The rest is taken care of by the control domain (Dom0), a spe-

FIGURE 1

XEN ARCHITECTURAL OVERVIEW



cialized guest granted elevated privileges by the hypervisor. This allows Dom0 to manage all other hardware in the system, as well as all other guests running on the hypervisor. It is also important to realize that Xen contains almost no hardware drivers, preventing code duplication with the drivers already present in OSes (See figure 1).

The Guests

Because Xen was designed back when x86 didn't have the hardware features it has now, it is able to offer several different virtualization modes. In the late 1990s, there were only two options to use virtualization on x86, both with very high overhead—full software emulation or binary translation. To overcome this, Xen took a new approach. We made the guest aware that it was running inside a virtualized environment and provided a whole new interface that removed the extra overhead. This led to what is known today as paravirtualization (PV), and replaced the following interfaces with PV-aware implementations:

- Disk and network
- Interrupts and timers
- Boot process, the guest starts directly in the mode it wishes to run (either 32 or 64 bits)
- Page tables
- Privileged instructions

Some of the interfaces listed above are easy to implement and are not intrusive regarding the guest OS, such as PV disks and nics. On the other hand, some are really intrusive, such as the replacement of the native page table implementation.

A couple of these interfaces are worth explaining in more detail, like the paravirtualization of interrupts. PV guests are not allowed to use native interrupts, so a new technique called event channels was introduced to replace them. Event channels use a single entry point into the guest kernel to inject interrupts, and there's a shared memory region between Xen and the guest to signal which event has fired. All interrupts are routed over this interface when running as a Xen PV guest.

Another important technique used by Xen guests is the hypercall page. This is a memory page in the guest OS that's filled by Xen and contains the hardware-specific hypercall implementations. Hypercalls are much like system calls between user-space and kernel-space in OSes, but in this case the hypercall is between

FIGURE 2

THE VIRTUALIZATION SPECTRUM

	DISK & NETWORK	INTERRUPTS & TIMERS	EMULATED MOTHERBOARD	PRIVILEGED INSTRUCTIONS & PAGE TABLES
HVM	🟥 VS	🟥 VS	🟨 VS	🟩 VH
HVM WITH PV DRIVERS	🟩 PV	🟥 VS	🟨 VS	🟩 VH
PVHVM	🟩 PV	🟩 PV	🟨 VS	🟩 VH
PVH	🟩 PV	🟩 PV	🟩 PV	🟩 VH
PV	🟩 PV	🟩 PV	🟩 PV	🟨 PV

🟥	POOR PERFORMANCE	VS	SOFTWARE VIRTUALIZATION
🟨	ROOM FOR IMPROVEMENT	VH	HARDWARE VIRTUALIZATION
🟩	OPTIMAL PERFORMANCE	PV	PARAVIRTUALIZED

the guest kernel and the hypervisor itself. Hypercalls are the only way a guest OS is able to communicate with the Xen hypervisor.

With the introduction of hardware virtualization extensions in x86 in 2005, Xen gained the ability to run unmodified guests in Hardware Virtual Machine (HVM) mode. This was a very important step because it allowed Xen to run guests without any PV-aware interfaces. In order to do this, a device model is needed (which usually runs in Dom0) that emulates the devices provided to the guest. All this emulation is handled by Qemu, which was adapted to work with Xen.

Using device models is expensive for both the Dom0 and the guest. Since each guest needs its own Qemu instance if all of them are running on Dom0, it can easily become a bottleneck in terms of CPU and memory usage. From a guest point of view, it also adds more overhead when compared to using PV interfaces only, because accesses to these emulated devices cause traps into Qemu.

While this separation between PV and HVM guests makes a clear cut, there have been several PV-specific improvements made available to HVM guests to obtain better performance. HVM guests can make use of PV disks and nics to boost IO throughput. When a guest makes use of those interfaces inside an HVM container, it is known as HVM with PV drivers in the Xen argot. But it doesn't stop here, since Xen 4.1, a HVM guest, can also use PV timers and PV IPIs to reduce even more emulation overhead. When a guest runs in this mode, it's known as PVHVM.

In general, HVM guests have better performance, especially regarding page table manipulation operations. The software page table manipulation used in PV guests is one of the main performance problems of pure PV guests. To improve this, a new mode has been recently introduced that allows PV guests to run inside HVM containers. This new mode is called PVH and makes use of the hardware virtualization extensions for the CPU and MMU, while using PV interfaces for the rest. **Figure 2** contains a table that shows the differences between the several guest modes supported by Xen.

NEW FEATURES IN XEN 4.4

Apart from the usual round of bug fixes and across-the-board improvements, the latest release of Xen includes several improvements worth mentioning. On the tools side, the default Xen toolstack (libxl) offers improved libvirt support. This lays the foundation for solid integration into any tools that can use libvirt, from GUI VM managers to cloud orchestration layers like CloudStack or OpenStack.

The Xen on ARM port also saw a number of improvements, as now Xen is able to both run on ARM 64-bit hardware and also support ARM 64-bit guests. The ARM ABI (the interface between the guests and the hypervisor) has also been declared stable, which means all changes will be done in a backwards compatible fashion, and ARM guests using the Xen 4.4 interface can rest assured this will continue to work on newer releases. Also this release added support for many new boards including Arndale, Calxeda ECX-2000, Applied Micro X-Gene Storm, TI QMAP5 and Allwinner A20/A30. There's already ongoing work to port FreeBSD as both a guest and a Dom0 for Xen on ARM.

One of the most interesting 4.4 features relevant to FreeBSD is the new virtualization mode called PVH. Xen 4.4 has experimental support for running non-privileged guests in PVH mode, with Dom0 PVH support coming in Xen 4.5. This new virtualization mode is very similar to PVHVM, a mode FreeBSD can run as of FreeBSD 10, but removes the need for any software emulation at all. This means there's no need to run a device model in order to run a PVH guest (something that is needed for PVHVM guests). Since PVH doesn't require a device model to run, it can also be used as Dom0, something that until now was only possible with pure PV guests.

PVH guests make heavy use of the hardware virtualization extensions found in current processors as it runs inside a HVM container and has access to a hardware-virtualized CPU and MMU, which means there are no PV interfaces for page table manipulation or privileged instruction execution. As said before, one of the hardest things about adding PV support into an OS is the fact that the virtual memory subsystem has to be rewritten or completely filled with hooks for Xen, which is both hard to design and hard to maintain. Apart from the simplification of the PV interface, PVH is also much faster regarding page table manipulations, since using the hardware-virtualized MMU is certainly faster than using the PV MMU. Although originally designed with performance in mind, PVH has turned out to be a very important feature for FreeBSD, greatly simplifying the path from PVHVM to being able to run as a Dom0. This means that FreeBSD can bypass all the pain of implementing PV MMU support without any downsides. Performance of PVH is better than the performance of pure PV, and it will also allow FreeBSD to run as Dom0. Due to all these benefits, it is very likely that the embryonic i386 PV port will be removed in favor of PVH.

XEN Support In FreeBSD 10

The 10 release cycle was quite interesting in terms of new Xen features. The FreeBSD/Xen port status in 9 included support for running as a uniprocessor 32-bit PV guest and as a HVM guest with PV drivers for both disk and nics. Most of the work during the last release focused on getting FreeBSD to run as a PVHVM guest and also improving the performance and features of the PV drivers. To run as a PVHVM guest, several under-the-hood improvements were done, and while those are not quite visible from a user point of view, they were crucial to getting us where we are now.

One of the first and most important improvements is the change in how event channel interrupts are injected and handled in FreeBSD. In previous releases, events for all event channels were signaled to the guest via a single, global PCI interrupt. This is simple and works fine if the guest is only using event channels for a few PV disk and nic devices. However it scales poorly since all event channel processing is tied to a single interrupt, running on a single CPU. In order to solve this limitation, support for Xen's newer event delivery scheme was added to FreeBSD. Known as "vector callback," Xen allows the guest to allocate an IDT vector for each event channel. Since Xen can inject an IDT event to a specific CPU, this allows full distribution of interrupt load across all CPUs. It also makes it possible to efficiently implement new, per-cpu, device types that are paravirtualized.

Thanks to the introduction of the vector callback, it is possible to use the PV timer. This timer is implemented as a one-shot-per-cpu event timer that is set using hypercalls and is delivered to the guest using an event channel interrupt. This removes the overhead of using the emulated timers, which comes from the fact that you need to perform reads and writes to the emulated devices registers, which cause VMEXITs into Xen. A VMEXIT is an involuntary trap into the hypervisor that involves a context switch. As with any context switch, the saving and restoring of execution state is costly and should be avoided when possible. VMEXITs occur anytime a guest attempts a privileged

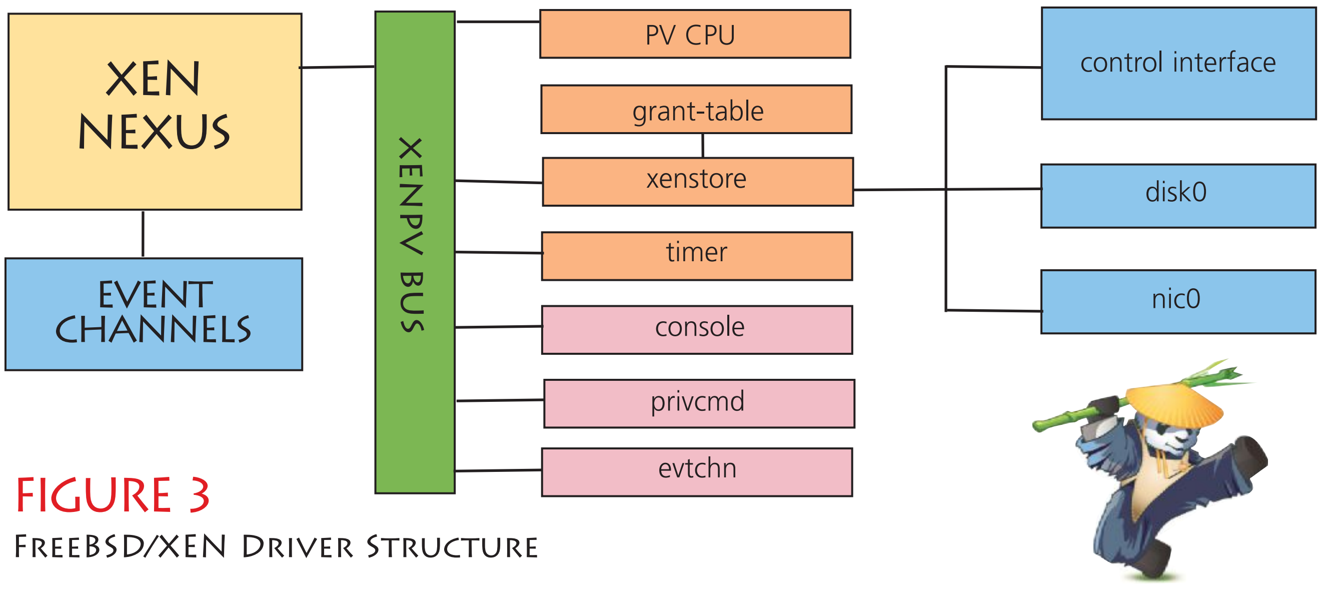


FIGURE 3

FreeBSD/XEN Driver Structure

operation: accessing certain registers, executing certain instructions, or accessing a memory location being managed by the hypervisor, like the memory of devices emulated by Xen.

By using the PV timer only one VMEXIT is taken when issuing the hypercall to set the timer. Xen also provides a shared memory region that contains a lot of time-related information. In addition to setting the PV timer, this information is also used to create a time counter and to provide a clock implementation for FreeBSD. As we can see, we can solve all the OS time-related needs using PV interfaces only.

Another improvement that was made to reduce the emulation overhead is to route inter processor interrupts (IPIs) over event channels. On native x86 hardware, IPIs are delivered using the local APIC, but again when running on a virtualized environment, accesses to the local APIC cause VMEXITs into Xen, which ideally we like to avoid. This is solved by using event channel interrupts to deliver these signals between processors, which greatly reduced the latency of IPIs. Here again we turn multiple VMEXITs into a single hypercall, reducing the emulation overhead.

PVH Support in Head

Since PVH is very similar to PVHVM in terms of the PV interfaces used, getting PVH support into FreeBSD was not a big deal. The main difference between PVHVM and PVH is that PVH uses the PV start sequence. This means that there are no emulated BIOS, and the guest is started by directly jumping into the kernel entry point with some basic page tables already set up by Xen on behalf of the guest. The FreeBSD i386 PV port already had this entry point; however most of the code there was not suitable for PVH, due to it being a fork of the i386 machdep code with the Xen-specific implementation hardcoded in it. We wanted to avoid this as much as possible, since the original idea was to have PVH support in the GENERIC kernel, so that the user would not be required to compile a custom kernel, and could use freebsd-update inside of a PVH guest without problems.

So the first step was to add the PV entry point and some elf notes that are used by Xen in order to know how to load the kernel. This is quite straightforward, and only requires a couple of assembly lines. After that, a specific Xen initialization function is called that sets the page tables as FreeBSD expects to find them (as set by the boot trampoline on native amd64). This function also initializes a couple of Xen-specific global variables with data provided by the hypervisor at boot time.

After that, the native FreeBSD initialization function is called, which has also been slightly tweaked to prevent it from using any emulated devices. Instead of adding a bunch of Xen-specific conditionals in common code, another approach was taken: strategically placed hooks that allow the runtime selection of the proper support code. By default, the hooks point to the native hardware implementation. But if Xen PVH mode is detected, the Xen code is activated. The early initialization hooks provide a dynamic way to change the clock source used during early bootstrap (8259 PIC vs. hypercall) and the method for fetching the hardware memory map (E820 vs. hypercall).

As PVH guests don't have a local APIC, the way to start the secondary CPUs is also different from native, so another hook had to be added to implement a PVH specific method to start them. On PVH, the way to start secondary CPUs is quite easy (less convoluted than on native I would say), and basically involves setting the initial values of the registers and starting the CPU. This removes the need for any kind of trampoline to set up the page tables, since the CPU is started directly with paging enabled.

Finally all the local APIC-related functions have been converted into hooks, which allows Xen-specific overrides for some of them. Most of them should never be called when running as a PVH domain (because there's no local APIC) so some asserts have been added to make sure no other subsystem tries to use them.

The addition of PVH support also included some major rework of how the Xen code is structured in FreeBSD. Until now, all top-level Xen devices were directly attached to the Nexus. With the introduction of PVH support, the code has been structured in a more hierarchical way by introducing a Xen-specific bus that all top-level Xen devices hang from. **Figure 3** shows how the new xenpv bus is structured, and how descendant devices are organized. Top-level Xen devices include the PV console, the PV timer, and xenstore. Xenstore is a very important component in PVH guests, because all the hardware description comes from it (there are no ACPI tables in PVH). As seen on the diagram, the PV disks and nics hang off xenstore.

Since there is no ACPI on PVH, FreeBSD was falling back to the legacy Nexus implementation. This is not desirable, because it attaches a bunch of buses not present on PVH. So a very simple Xen-specific Nexus is provided in the PVH case that fills this role. Finally, two device drivers were added: the evtchn device, which allows user-space applications to bind event channels, and the privcmd device, present only when running as Dom0, that is used to send management commands from user-space to the hypervisor.

The Future

With PVH support already merged in HEAD, the work has now shifted to PVH Dom0 support. Running as Dom0 is quite different from running as a guest. One of the main differences is that Dom0 has access to the physical hardware, and so it must manage it. But it's not identical to running on bare metal. Two examples of the many differences are physical hardware interrupts delivered via Xen event channels, and the guest's need to set up its physical interrupts using hypercalls.

This might sound very convoluted, but it's actually not that difficult. Xen provides hypercalls that allow Dom0 to set up IO APIC, MSIs and MSI-Xs interrupts in an easy way and without having to deal with the underlying hardware. Again, all of this has been implemented with hooks into the existing code, using the infrastructure provided by newbus, which makes it trivial to replace certain methods with their Xen counterparts. Thanks to this interface, no changes are required to drivers at all.

Regarding the Xen-specific code, some modifications are needed to xenstore in order to run as Dom0. Xenstore is an information storage space shared between domains maintained by the xenstored application. When running as a guest, xenstore contains the hardware description for the domain and is always accessible, but that's not the case for Dom0, since xenstore has not yet been started.

So in Dom0 we have to allow the boot process to skip the xenstore initialization until the daemon has actually been started, and we have to prevent any Xen kernel driver from trying to use xenstore until it is actually running. That's solved by not attaching the xenstore bus until the process has been launched, and thanks to the hierarchical structure of the Xen components in FreeBSD, it's easy to accomplish: just hold off attaching any driver that hangs off xenstore until it is actually initialized.

FreeBSD already has the necessary drivers for providing network and disk services to guests. Thanks to that, there's not much work to do. Some of those backends haven't seen much use, so they will probably need some fine tuning, but that's much less work than actually writing them from scratch.

Dom0 support is still in its early stages, but overall it looks very promising. FreeBSD is heavily focused on performance, especially IO performance, which is exactly what Dom0 needs, since it usually runs a bunch of PV backends to serve requests from guests. Having cutting-edge features like ZFS and Netmap inside the kernel is certainly going to make FreeBSD a very interesting OS within the Xen Project ecosystem. ●

Roger Pau Monné is a Software Engineer at Citrix and a FreeBSD developer. He usually contributes to the Xen Project and the FreeBSD/Xen port, and right now is mainly focused on getting stable PVH support in both projects.

Using bhyve FOR FREEBSD DEVELOPMENT

by John Baldwin

One of the exciting new features in FreeBSD 10.0 is the **bhyve hypervisor**. *Hypervisors and virtual machines are used in a wide variety of applications. This article focuses on using bhyve as a tool for aiding development of FreeBSD itself. Not all of the details covered are specific to FreeBSD development, however, and many may prove useful for other applications.*

✤ Note that the bhyve hypervisor is under constant development and some of the features described have been added since FreeBSD 10.0 was released. Most of these features should be present in FreeBSD 10.1.

The Hypervisor

The bhyve hypervisor requires a 64-bit x86 processor with hardware support for virtualization. This requirement allows for a simple, clean hypervisor implementation, but it does require a fairly recent processor. The current hypervisor requires an Intel processor, but there is an active development branch with support for AMD processors.

The hypervisor itself contains both user and kernel components. The kernel driver is contained in the `vmm.ko` module and can be loaded either at boot from the boot loader or at runtime. It must be loaded before any guests can be created. When a guest is created, the kernel driver creates a device file in `/dev/vmm` which is used by the user programs to interact with the guest.

The primary user component is the http://www.freebsd.org/cgi/man.cgi?query=bhyve&sektion=8&manpath=FreeBSD+10.0-RELEASE bhyve(8) program. It constructs the emulated device tree in the guest and provides the implementation for most of the emulated devices. It also calls the kernel driver to execute the guest. Note that the guest always executes inside the driver itself, so guest execution time in the host is counted as system time in the bhyve process.

Currently, bhyve does not provide a system firmware interface to the guest (neither BIOS nor UEFI). Instead, a user program running on the host is used to perform boot time operations including loading the guest operating system kernel into the guest's memory and setting the initial guest state so that the guest begins execution at the kernel's entry point. For FreeBSD guests, the http://www.freebsd.org/cgi/man.cgi?query=bhyveload&sektion=8&manpath=FreeBSD+10.0-RELEASE bhyveload(8) program can be used to load the kernel and prepare the guest for execution. Support for some other operating systems is available via the "https://github.com/grehan-freebsd/grub2-bhyve"grub2-bhyve program which is available via the "http://www.freshports.org/sysutils/grub2-bhyve/"sysutils/grub2-bhyve port or as a prebuilt package.

The bhyveload(8) program in FreeBSD 10.0 only supports 64-bit guests. Support for 32-bit guests will be included in FreeBSD 10.1.

Network Setup

The network connections between the guests and the host can be configured in several ways. Two different setups are described below, but they are not the only possible configurations.

The only guest network driver currently supported by bhyve is the "http://docs.oasis-open.org/virtio/virtio/v1.0/virtio-v1.0.html"VirtIO network interface driver. Each network interface exposed to the guest is associated with a http://www.freebsd.org/cgi/man.cgi?query=tap%284%29 tap(4) Interface in the host. The tap(4) driver allows a user program to inject packets into the network stack and accept packets from the network stack. By design, each tap(4) interface will only pass traffic if it is opened by a user process and it is administratively enabled via "http://www.freebsd.org/cgi/man.cgi?query=ifconfig%288%29" ifconfig(8). As a result, each tap(4) interface must be explicitly enabled each time a guest is booted. This can be inconvenient for frequently restarted guests. The tap(4) driver can be changed to automatically enable an interface when it is opened by a user process by setting the `net.link.tap.up_on_open` sysctl to 1.

Bridged Configuration

One simple network setup bridges the guest network interfaces directly onto a network to which the host is connected. On the host, a single "http://www.freebsd.org/cgi/man.cgi?query=if_bridge%284%29"if_bridge (4) interface is created. The tap(4) interfaces for the guest are added to the bridge along with the network interface on the host that is attached to the desired network. Example 1 connects a guest using `tap0` to a LAN on the host's `re0` interface:

```
# ifconfig bridge0 create
# ifconfig bridge0 addm re0
# ifconfig bridge0 addm tap0
# ifconfig bridge0 up
```

Example 1: Manually Connecting a Guest to the Host's LAN

The guest can then configure the VirtIO network interface bound to `tap0` for the LAN on the host's `re0` interface using DHCP or a static address.

The `/etc/rc.d/bridge` script allows bridges to be configured during boot automatically by variables in `/etc/rc.conf`. The `autobridge_interfaces` variable lists the bridge interfaces to configure. For each bridge interface, an `autobridge_<name>` variable lists other network interfaces that should be added as bridge members. The list can include shell globs to match multiple interfaces. Note that `/etc/rc.d/bridge` will not create the named bridge interfaces. They should be created by listing them in the `cloned_interfaces` variable along with the desired tap(4) interfaces. Example 2 lists the `/etc/rc.conf` settings to create three tap(4) interfaces bridged to a local LAN on the host's `re0` interface.

```
/etc/rc.conf
autobridge_interfaces="bridge0"
autobridge_bridge0="re0 tap*"
cloned_interfaces="bridge0 tap0 tap1 tap2"
ifconfig_bridge0="up"
```

Example 2: Bridged Configuation

Private Network with NAT

A more complex network setup creates a private network on the host for the guests and uses network address translation (NAT) to provide limited access from guests to other networks.

This may be a more appropriate setup when the host is mobile or connects to untrusted networks.

This setup also uses an if_bridge(4) interface, but only the tap(4) interfaces used by guests are added as members to the bridge. The bridge members are assigned addresses from a private subnet. The bridge interface is assigned an address from the private subnet as well to connect the bridge to the host's network stack. This allows the guests and host to communicate over the private subnet used by the bridge.

The host acts as a router for the guests to permit guest access to remote systems. IP forwarding is enabled on the host and guest connections are translated via "http://www.freebsd.org/cgi/man.cgi?query=natd%288%29" natd(8) The guests use the host's address in the private subnet as their default route.

Example 3 lists the `/etc/rc.conf` settings to create three tap(4) interfaces and a bridge interface using the 192.168.16.0/24 subnet. It also translates network connections over the host's `wlan0` interface using natd(8).

```
/etc/rc.conf
autobridge_interfaces="bridge0"
autobridge_bridge0="tap*"
cloned_interfaces="bridge0 tap0 tap1 tap2"
ifconfig_bridge0="inet 192.168.16.1/24"
gateway_enable="YES"
natd_enable="YES"
natd_interface="wlan0"
firewall_enable="YES"
firewall_type="open"
```

Example 3: Private Network Configuration

```
/usr/local/etc/dnsmasq.conf
domain-needed
bogus-priv
interface=bridge0
dhcp-range=192.168.16.10,192.168.16.200,12h
```

Example 4: Enabling dnsmasq's DNS and DHCP Servers

Using dnsmasq with a Private Network

The private network from the previous example works well, but it is a bit tedious to work with. Guests must statically configure their network interfaces, and network connections between guests and the host must use hardcoded IP addresses. The "http://www.thekelleys.org.uk/dnsmasq/doc.html"dnsmasq utility can alleviate much of the tedium. It can be installed via the "http://www.freshports.org/dns/dnsmasq/ "dns/dnsmasq port or as a prebuilt package.

The dnsmasq daemon provides a DNS forwarding server as well as a DHCP server. It can serve local DNS requests to map the hostnames of its DHCP clients to the addresses it assigns to the clients. For the private network setup, this means that each guest can use DHCP to configure its network interface. In addition, all of the guests and the host can resolve each guest's hostname.

The dnsmasq daemon is configured by settings in the `/usr/local/etc/dnsmasq.conf` configuration file. A sample configuration file is installed by the port by default. The configuration file suggests enabling the **`domain-needed`** and **`bogus-priv`** settings for the DNS server to avoid sending useless DNS requests to upstream DNS servers. To enable the DHCP server, **`interface`** must be set to the network interface on the host where the server should run, and **`dhcp-range`** must be set to configure the range of IP addresses that can be assigned to guests.

Example 4 instructs the dnsmasq daemon to run a DHCP server on the `bridge0` interface and assign a subset of the 192.168.16.0/24 subnet to guests.

In addition to providing local DNS names for DHCP clients, dnsmasq also provides DNS names for any entries in `/etc/hosts` on the host. An entry to `/etc/hosts` that maps the IP address assigned to `bridge0` to a hostname (e.g.,"host") will allow guests to use that hostname to contact the host.

The last thing remaining is to configure the host machine to use dnsmasq's DNS server. Allowing the host to use dnsmasq's DNS server allows the host to resolve the name of each guest. The dnsmasq daemon can use "http://www.freebsd.org/cgi/man.cgi?query=resolvconf%288%29"resolvconf(8) to seamlessly handle updates to the host's DNS configuration provided by DHCP or VPN clients. This is implemented by resolvconf(8) updating two configuration files that are read by dnsmasq each time the host's DNS configuration changes. Finally, the host should always use dnsmasq's DNS server and rely on it to forward requests to other upstream DNS servers. Enabling all this requires changes to both dnsmasq's configuration file and `/etc/resolvconf.conf`. More details about configuring resolvconf(8) can be found in "http://www.freebsd.org/cgi/man.cgi?query=resolvconf.conf%285%29"resolvconf.conf(5). Example 5 gives the changes to both files to use dnsmasq as the host's name resolver.

Running Guests via vmrun.sh

Executing a guest requires several steps. First, any state from a previous guest using the same name must be cleared before a new guest can begin. This is done by passing the `--destroy` flag to bhyvectl(8). Second, the guest must be created and the guest's kernel must be loaded into its address space by bhyveload(8) or grub2-bhyve. Finally, the bhyve(8) program is used to create virtual devices and provide runtime support for guest execution. Doing this all by hand for each guest invocation can be a bit tedious, so FreeBSD ships with a wrapper script for FreeBSD guests: `/usr/share/examples/bhyve/vmrun.sh`.

The vmrun.sh script manages a simple FreeBSD guest. It performs the three steps above in a loop so that the guest restarts after a reboot similar to real hardware. It provides a fixed set of virtual devices to the guest including a network interface backed by a tap(4) interface, a local disk backed by a disk image, and an optional second disk backed by an install image. To make guest installations easier, vmrun.sh checks the provided disk image for a valid boot sector. If none is found, it instructs bhyveload(8) to boot from the install image, otherwise it boots from the disk image. In FreeBSD 10.1 and later, vmrun.sh will terminate its loop when the guest requests soft off via ACPI.

The simplest way to bootstrap a new FreeBSD guest is to install the guest from an install ISO image. For a FreeBSD guest running 9.2 or later, the standard install process can be used by using the normal install ISO as the optional install image passed to vmrun.sh. FreeBSD 8.4 also works as a bhyve guest. However, its installer does not fully support VirtIO block devices, so the initial install must be performed manually using steps similar to those from the "https://wiki.freebsd.org/RootOnZFS" RootOnZFS guide. Example 6 creates a 64-bit guest named "vm0" and boots the install CD for FreeBSD 10.0-RELEASE. Once the guest has been installed, the `-I` argument can be dropped to boot the guest from the disk image.

The vmrun.sh script runs bhyve(8) synchronously and uses its standard file descriptors as the backend of the first serial port assigned to the guest. This serial port is used as the system console device for FreeBSD guests. The simplest way to run a guest in the background using vmrun.sh is to use a tool such as "http://www.freshports.org/sysutils/screen"screen or "http://www.freshports.org/sysutils/tmux"tmux.

FreeBSD 10.1 and later treat the `SIGTERM` signal sent to bhyve(8) as a virtual power button. If the guest supports ACPI, then sending `SIGTERM` interrupts the guest to request a clean shutdown. The guest should then initiate an ACPI soft-off which will terminate the vmrun.sh loop. If the guest does not respond to `SIGTERM`, the guest can still be forcefully terminated from the host via `SIGKILL`. If the guest does not support ACPI, then `SIGTERM` will immediately terminate the guest.

The vmrun.sh script accepts several different arguments to control the behavior of bhyve(8) and bhyveload(8), but these arguments only permit enabling a subset of the features supported by these programs. To control all available features or use alternate virtual device configurations (e.g., multiple virtual drives or network interfaces), either invoke bhyveload(8) and bhyve(8) manually or use vmrun.sh as the basis of a custom script.

/usr/local/etc/dnsmasq.conf

```
conf-file=/etc/dnsmasq-conf.conf
resolv-file=/etc/dnsmasq-resolv.conf
```

/etc/resolvconf.conf

```
name_servers=127.0.0.1
dnsmasq_conf=/etc/dnsmasq-conf.conf
dnsmasq_resolv=/etc/dnsmasq-resolv.conf
```

Example 5: Use dnsmasq as the Host's Resolver

Configuring Guests

FreeBSD guests do not require extensive configuration settings to run, and most settings can be set by the system installer. However, there are a few conventions and additional settings which can be useful.

Out of the box, FreeBSD releases prior to 9.3 and 10.1 expect to use a video console and keyboard as the system console. As such, they do not enable a login prompt on the serial console.

```
# mkdir vm0
# truncate -s 8g vm0/disk.img
# sh /usr/share/examples/bhyve/vmrun.sh -t tap0 -d vm0/disk.img \
   -I FreeBSD-10.0-RELEASE-amd64-disc1.iso vm0
```

Example 6: Creating a FreeBSD/amd64 10.0 Guest

“Once kgdb(1) attaches to the remote target, it can be used to debug the guest kernel”

```
/etc/rc.conf
   hostname="vm0"
   ifconfig_vtnet0="DHCP"
   sshd_enable="YES"
   dumpdev="AUTO"
   sendmail_enable="NONE"
```

Example 7: Simple FreeBSD Guest Configuration

```
> cd ~/work/freebsd/head/sys/amd64/compile/GUEST
> make install DESTDIR=~/bhyve/vm0/host KMODOWN=john
...
> cd ~/bhyve
> sudo sh vmrun.sh -t tap0 -d vm0/disk.img -H vm0/host vm0
...
OK unload
OK load host0:/boot/kernel/kernel
host0:/boot/kernel/kernel text=0x523888 data=0x79df8+0x10e2e8 syms=[0x8+0x9fb58+0x8+0xbaf41]
OK boot
...
Copyright (c) 1992-2014 The FreeBSD Project.
Copyright (c) 1979, 1980, 1983, 1986, 1988, 1989, 1991, 1992, 1993, 1994
        The Regents of the University of California. All rights reserved.
FreeBSD is a registered trademark of The FreeBSD Foundation.
FreeBSD 11.0-CURRENT #6 r261528M: Fri Feb  7 09:55:45 EST 2014
    john@pippin.baldwin.cx:/usr/home/john/work/freebsd/head/sys/amd64/compile/GUEST amd64
```

Example 8: Booting a Kernel from the Host.

A login prompt should be enabled on the serial console by editing `/etc/ttys` and marking the `ttyu0` terminal "on." Note that this can be done from the host after the install has completed by mounting the disk image on the host using "http://www.freebsd.org/cgi/man.cgi?query=mdconfig%288%29"mdconfig(8).

Note: Be sure the guest is no longer accessing the disk image before mounting its filesystem on the host to avoid data corruption.

If a guest requires network access, it will require configuration similar to that of a normal host. This includes configuring the guest's network interface `vtnet0` and assigning a hostname. A useful convention is to reuse the name of the guest("vm0" in Example 6) as the hostname. The "http://www.freebsd.org/cgi/man.cgi?query=sendmail%288%29" sendmail(8) daemon may hang attempting to resolve the guest's hostname during boot. This can be worked around by completely disabling sendmail(8) in the guest. Finally, most guests with network access will want to enable remote logins via "http://www.freebsd.org/cgi/man.cgi?query=sshd%288%29"sshd(8). Example 7 lists the `/etc/rc.conf` file for a simple FreeBSD guest.

Using a bhyve Guest as a Target

One way bhyve can be used while developing FreeBSD is to allow a host to debug a guest as if the guest were a remote target. Specifically, a test kernel can be built on the host, booted inside of the guest, and debugged from the host using http://www.freebsd.org/cgi/man.cgi?query=kgdb%281%29 kgdb(1).

Once a guest is created and configured and a test kernel has been built on the host, the next step is to boot the guest with the test kernel. The traditional method is to install the kernel into the guest's filesystem either by exporting the build directory to the guest via NFS, copying the kernel into the guest over the network, or mounting the guest's filesystem on the host directly via mdconfig(8). An alternate method is available via bhyveload(8) which is similar to booting a test machine over the network.

Using bhyveload(8)'s Host Filesystem

The bhyveload(8) program allows a directory on the host's filesystem to be exported to the loader environment. This can be used to load a kernel and modules from a host filesystem rather than the guest's disk image. The directory on the host's filesystem is passed to bhyveload(8) via the `-h` flag. The bhyveload(8) program exports a `host0:` device to the loader environment. The path passed to the `host0:` device in the loader environment is appended to the configured directory to generate a host pathname. Note that the directory passed to bhyveload(8) must be an absolute pathname. The vmrun.sh script in FreeBSD 10.1 and later allow the directory to be set via the `-H` argument. The script will convert a relative pathname to an absolute pathname before passing it to bhyveload(8).

Booting a test kernel from the host inside of the guest involves the above three steps: Example 8 installs a kernel with the kernel config GUEST into a `host` directory for the guest "vm0". It uses vmrun.sh's `-H` argument to specify the host directory passed to bhyveload(8). It also shows the commands used at the loader prompt to boot the test kernel.

The guest can be configured to load a kernel from the `host0:` filesystem on the next boot using "http://www.freebsd.org/cgi/man.cgi?query=nextboot%288%29" nextboot(8). To boot the `host0:/boot/kernel/kernel` kernel, run `nextboot -e bootfile=host0:/boot/kernel/kernel` before rebooting. Note that this will not adjust

3 STEPS for booting a test kernel

- Install the kernel into the directory on the host by setting the `DESTDIR` variable to the directory when invoking `make install` or `make installkernel`. A non-root user with write access to the directory can perform this step directly by setting the `KMODOWN` make variable to the invoking user.
- Pass the directory's path to bhyveload(8) either via the `-h` flag to bhyveload(8) or the `-H` flag to vmrun.sh.
- Explicitly load the new kernel at the bhyveload(8) prompt via the loader path `host0:/boot/kernel/kernel`.

ISILON The industry leader in Scale-Out Network Attached Storage (NAS)

Isilon is deeply invested in advancing FreeBSD performance and scalability. We are looking to hire and develop FreeBSD committers for kernel product development and to improve the Open Source Community.

We're Hiring!

With offices around the world, we likely have a job for you! Please visit our website at http://www.emc.com/careers or send direct inquiries to annie.romas@emc.com.

EMC² ISILON

```
> sudo sh vmrun.sh -t tap0 -d vm0/disk.img -H vm0/host -g 1234 vm0
...
OK load host0:/boot/kernel/kernel
host0:/boot/kernel/kernel text=0x523888 data=0x79df8+0x10e2e8 syms=[0x8+0x9fb58+0x8+0xbaf41]
OK boot
Booting...
GDB: debug ports: bvm
GDB: current port: bvm
...
root@vm0:~ # sysctl debug.kdb.enter=1
debug.kdb.enter: 0KDB: enter: sysctl debug.kdb.enter
[ thread pid 693 tid 100058 ]
Stopped at      kdb_sysctl_enter+0x87:  movq    $0,kdb_why
db> gdb
(ctrl-c will return control to ddb)
Switching to gdb back-end
Waiting for connection from gdb
-> 0
root@vm0:~ #
```

Example 9: Using kgdb(1) with bvmdebug: In the Guest

```
> cd ~/work/freebsd/head/sys/amd64/compile/GUEST
> kgdb kernel.debug
...
(kgdb) target remote localhost:1234
Remote debugging using localhost:1234
warning: Invalid remote reply:
kdb_sysctl_enter (oidp=<value optimized out>, arg1=<value optimized out>,
   arg2=1021, req=<value optimized out>) at ../../../kern/subr_kdb.c:446
446                     kdb_why = KDB_WHY_UNSET;
Current language:  auto; currently minimal
(kgdb) c
Continuing.
```

Example 10: Using kgdb(1) with bvmdebug: On the Host

the module path used to load kernel modules, so it only works with a monolothic kernel.

Using bhyve(8)'s Debug Port

The bhyve(8) hypervisor provides an optional debug port that can be used by the host to debug the guest's kernel using kgdb(1). To use this feature, the guest kernel must include the **bvmdebug** device driver, the **KDB** kernel debugger, and the **GDB** debugger backend. The debug port must also be enabled by passing the **-g** flag to bhyve(8). The flag requires an argument to specify the local TCP port on which bhyve(8) will listen for a connection from kgdb(1). The vmrun.sh script also accepts a **-g** flag which is passed through to bhyve(8).

When the guest boots, its kernel will detect the debug port as an available GDB backend automatically. To connect kgdb(1) on the host to the guest, first enter the kernel debugger by setting the **debug.kdb.enter** system control node to a non-zero value. At the debugger prompt, invoke the **gdb** command. On the host, run kgdb(1) using the guest's kernel as the kernel image. The **target remote** command can be used to connect to the TCP port passed to bhyve(8). Once kgdb(1) attaches to the remote target, it can be used to debug

"The bhyve(8) hypervisor provides an optional debug port that can be used by the host to debug the guest's kernel using kgdb(1)."

the guest kernel. Examples 9 and 10 demonstrate these steps using a guest kernel built on the host.

Using kgdb(1) with a Virtual Serial Port
A serial port can also be used to allow the host to debug the guest's kernel. This can be done by loading the "http://www.freebsd.org/cgi/man.cgi?query=nmdm%284%29"nmdm(4) driver on the host and using a nmdm(4) device for the serial port used for debugging.

To avoid spewing garbage on the console, connect the nmdm(4) device to the second serial port. This is enabled in the hypervisor by passing `-l com2,/dev/nmdm0B` to bhyve(8). The guest must be configured to use the second serial port for debugging by setting the kernel environment variable `hint.uart.1.flags=0x80` from bhyveload(8). The kgdb(1) debugger on the host connects to the guest by using `target remote /dev/nmdm0A`.

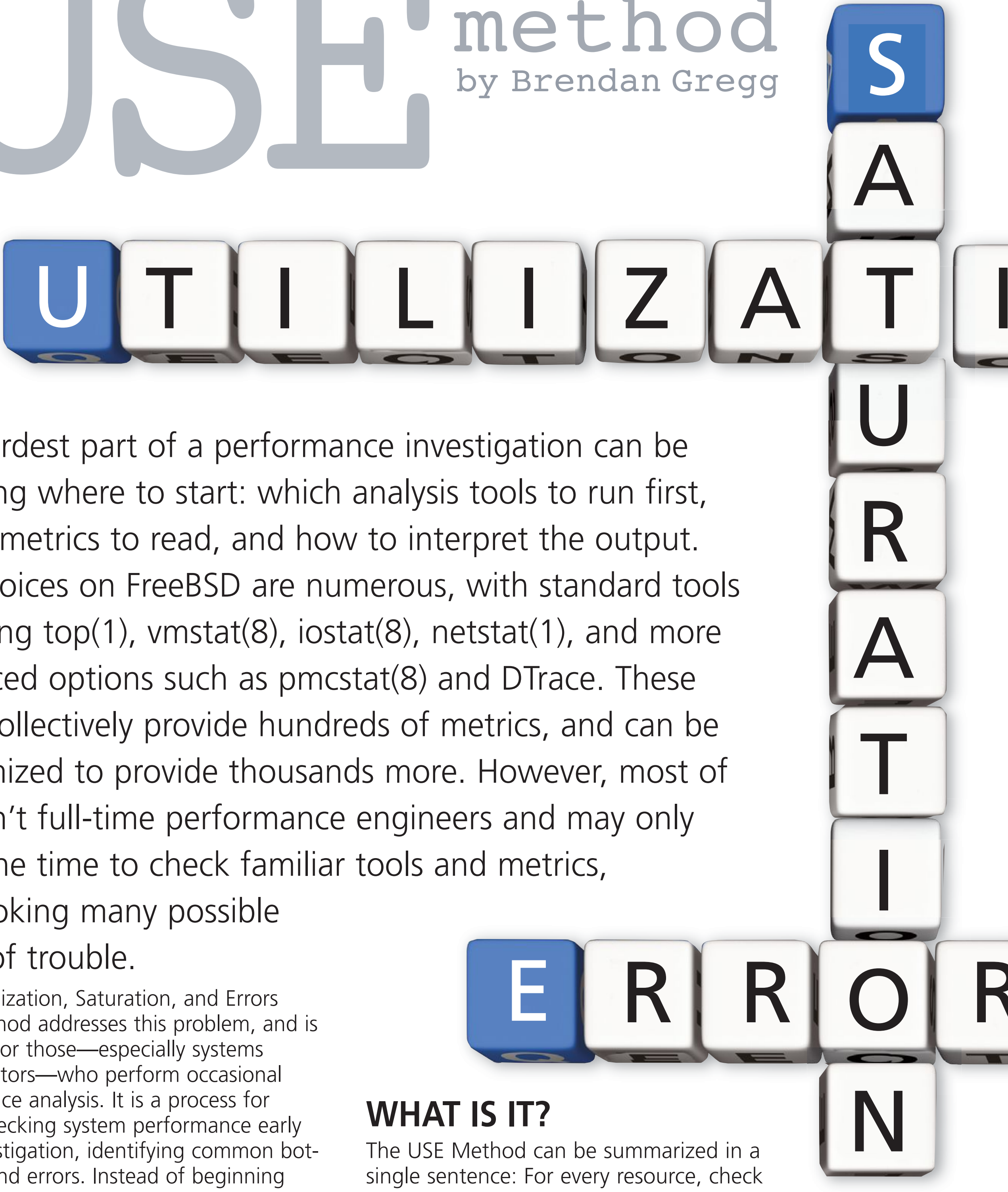
Conclusion

The bhyve hypervisor is a nice addition to a FreeBSD developer's toolbox. Guests can be used both to develop new features and to test merges to stable branches. The hypervisor has a wide variety of uses beyond developing FreeBSD as well. ●

John Baldwin joined the FreeBSD Project as a committer in 1999. He has worked in several areas of the system, including SMP infrastructure, the network stack, virtual memory, and device driver support. John has served on the Core and Release Engineering teams and organizes an annual FreeBSD developer summit each spring.

the USE method

by Brendan Gregg



The hardest part of a performance investigation can be knowing where to start: which analysis tools to run first, which metrics to read, and how to interpret the output. The choices on FreeBSD are numerous, with standard tools including top(1), vmstat(8), iostat(8), netstat(1), and more advanced options such as pmcstat(8) and DTrace. These tools collectively provide hundreds of metrics, and can be customized to provide thousands more. However, most of us aren't full-time performance engineers and may only have the time to check familiar tools and metrics, overlooking many possible areas of trouble.

The Utilization, Saturation, and Errors (USE) method addresses this problem, and is intended for those—especially systems administrators—who perform occasional performance analysis. It is a process for quickly checking system performance early in an investigation, identifying common bottlenecks and errors. Instead of beginning with the tools and their statistics, the USE method begins with the questions we'd like answered. That way, we ensure that we don't overlook things due to a lack of familiarity with tools or a lack of the tools themselves.

WHAT IS IT?

The USE Method can be summarized in a single sentence: For every resource, check Utilization, Saturation, and Errors. Resources are any hardware in the datapath, including CPUs, memory, storage devices, network interfaces, controllers, and busses. Software resources, such as mutex locks, thread pools, and resource control limits, can also be studied. Utilization is usually the portion of time a resource was busy servicing work, with the exception of storage devices where utilization can also mean used capacity. Saturation is the degree to which a resource is overloaded with work, such as the length of a backlog queue.

Usually individual resource statistics also need to be investigated. Averaging CPU utilization across 32 CPUs can hide individual CPUs that are running at 100%, and the same can also be true for disk utilization. Identifying these is the target of the USE method, as they can become systemic bottlenecks.

I employed the USE method recently during a performance evaluation, where our system was benchmarked and compared to a similar one running Linux. The potential customer was dissatisfied with how ZFS performed in comparison, despite it being backed by SSDs in our system. Having worked both ZFS and SSD pathologies before, I knew that this could take days to unravel, and would involve studying complex behaviors of ZFS internals and SSD firmware. Instead of heading in those directions, I began by performing a quick check of system health using the USE method, while the prospect's benchmark was running. This immediately identified that the CPUs were at 100% utilization, while the disks (and ZFS) were completely idle. This helped reset the evaluation and shifted blame to where it was due (not ZFS), saving both my time and theirs.

Employing the USE method involves developing a checklist of combinations of resource and USE statistics, along with the tools in your current environment for observing them. I've included examples here, developed on FreeBSD 10 alpha 2, for both hardware and some software resources. This includes some new DTrace one-liners for exposing various metrics. You may copy these to your own company documentation (wiki), and enhance them with any additional tools you use in your environment.

These checklists include a terse summary of the targets, tools, and metrics to study. In some cases, the metric is straightforward reading from command line tools. Many others require some math, inference, or quite a bit of digging. This will hopefully get easier in the future, as tools are developed to provide USE method metrics more easily.

[1] e.g., using per-CPU run queue length as the

2014 EuroBSD Conference

in Sofia, Bulgaria!

InterExpo Congress Center,
147, Tsarigradsko shose blvd, Sofia, Bulgaria.

September 27-28th
FOR MAIN CONFERENCE

September 25-26th
FOR TUTORIALS

Call for Papers
Send Your Proposal to:
submission@eurobsdcon.org

Become a Sponsor
Contact Us Via:
oc-2014@eurobsdcon.org

EuroBSDcon is the premier European conference on the Open Source BSD operating systems attracting highly skilled engineering professionals, software developers, computer science students, professors, and users from all over the world. The goal of EuroBSDcon is to exchange knowledge about the BSD operating systems, facilitate coordination and cooperation among users and developers.

http://eurobsdcon.org

- **Talks & Schedule**
  More Information on the Conference Planning
- **Travel & Stay**
  Upcoming Details on Transportation and Hotels
- **Venue & Location**
  Information and Directions to the Conference Location
- **Spouses Track and Trip**
  Visit the Historic City of Plovdiv, and Tour Rila Monastery

PHYSICAL RESOURCES

COMPONENT	TYPE	METRIC
CPU	utilization	system-wide: vmstat 1, “us” + “sy”; per-cpu: vmstat -P; per-process: top, “WCPU”
CPU	saturation	system-wide: uptime, “load averages” > CPU count; vmstat 1, “procs:r” > CPU cou
CPU	errors	dmesg; /var/log/messages; pmcstat for PMC and whatever error counters are suppo
Memory capacity	utilization	system-wide: vmstat 1, “fre” is main memory free; top, “Mem:”; per-process: top - “RSS” is resident set size (Kbytes), “VSZ” is virtual memory size (Kbytes)
Memory capacity	saturation	system-wide: vmstat 1, “sr” for scan rate, “w” for swapped threads (was saturated
Memory capacity	errors	physical: dmesg?; /var/log/messages?; virtual: DTrace failed malloc()s
Network Interfaces	utilization	system-wide: netstat -i 1, assume one very busy interface and use input/output “byt
Network Interfaces	saturation	system-wide: netstat -s, for saturation related metrics, eg netstat -s l egrep ‘retransl
Network Interfaces	errors	system-wide: netstat -s l egrep ‘badlchecksum’, for various metrics; per-interface: ne
Storage device I/O	utilization	system-wide: iostat -xz 1, “%b”; per-process: DTrace io provider, eg, iosnoop or iotc
Storage device I/O	saturation	system-wide: iostat -xz 1, “qlen”; DTrace for queue duration or length [4]
Storage device I/O	errors	DTrace io:::done probe when /args[0]->b_error != 0/
Storage capacity	utilization	file systems: df -h, “Capacity”; swap: swapinfo, “Capacity”; pstat -T, also shows sw
Storage capacity	saturation	not sure this one makes sense - once its full, ENOSPC
Storage capacity	errors	DTrace; /var/log/messages file system full messages
Storage controller	utilization	iostat -xz 1, sum IOPS & tput metrics for devices on the same controller, and compa
Storage controller	saturation	check utilization and DTrace and look for kernel queueing
Storage controller	errors	DTrace the driver
Network controller	utilization	system-wide: netstat -i 1, assume one busy controller and examine input/output “by
Network controller	saturation	see network interface saturation
Network controller	errors	see network interface errors
CPU interconnect	utilization	pmcstat (PMC) for CPU interconnect ports, tput / max
CPU interconnect	saturation	pmcstat and relevant PMCs for CPU interconnect stall cycles
CPU interconnect	errors	pmcstat and relevant PMCs for whatever is available
Memory interconnect	utilization	pmcstat and relevant PMCs for memory bus throughput / max, or, measure CPI and
Memory interconnect	saturation	pmcstat and relevant PMCs for memory stall cycles
Memory interconnect	errors	pmcstat and relevant PMCs for whatever is available
I/O interconnect	utilization	pmcstat and relevant PMCs for tput / max if available; inference via known tput fror
I/O interconnect	saturation	pmcstat and relevant PMCs for I/O bus stall cycles
I/O interconnect	errors	pmcstat and relevant PMCs for whatever is available

saturation metric: dtrace -n ‘profile-99 { @[cpu] = lquantize(`tdq_cpu[cpu].tdq_load, 0, 128, 1); } tick-1s { printa(@); trunc(@); }’ where > 1 is saturation. If you’re using the older BSD scheduler, profile runq_length[]. There are also the sched:::load-change and other sched probes.

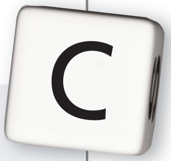
[2] For this metric, we can use time spent in TDS_RUNQ as a per-thread saturation (latency) metric. Here is an (unstable) fbt-based one-liner: dtrace -n ‘fbt::tdq_runq_add:entry { ts[arg1] = timestamp; } fbt::choosethread:return /ts[arg1]/ { @[stringof(args[1]->td_name), “runq (ns)”] = quantize(timestamp - ts[arg1]); ts[arg1] = 0; }’. This would be better (stability) if it can be rewritten to use the DTrace sched probes. It would also be great if there were simply high resolution thread state times in struct rusage or rusage_ext, eg, cumulative times for each state in td_state and more, which would make reading this metric easier and have lower overhead.

[3] e.g., for swapping: dtrace -n ‘fbt::cpu_thread_swapin:entry, fbt::cpu_thread_swapout:entry { @[probefunc, stringof(args[0]->td_name)] = count(); }’ (NOTE, I would trace vm_thread_swapin() and vm_thread_swapout(), but their probes don’t exist). Tracing paging is tricker until the vminfo provider is added; you could try tracing from swap_pager_putpages() and swap_pager_getpages(), but I didn’t see an easy way to walk back to a thread struct; another approach may be via vm_fault_hold(). Good luck. See thread states [2], which could make this much easier.

[4] e.g., sampling GEOM queue length at 99 Hertz: dtrace -n ‘profile-99 { @[“geom qlen”] = lquantize(`g_bio_run_down.bio_queue_length, 0, 256, 1); }’

[5] This approach is different from storage

for weighted and recent usage; per-kernel-process: top -S, “WCPU”
nt; per-cpu: DTrace to profile CPU run queue lengths [1]; per-process: DTrace of scheduler events [2]
rted (eg, thermal throttling)
o res, “RES” is resident main memory size, “SIZE” is virtual memory size; ps -auxw,
, may not be now); swapinfo, “Capacity” also for evidence of swapping/paging; per-process: DTrace [3]
tes” / known max (note: includes localhost traffic); per-interface: netstat -I interface 1, input/output “bytes” / known max
droplout-of-orderlmemory problemsloverflow’; per-interface: DTrace
etstat -i, “Ierrs”, “Oerrs” (eg, late collisions), “Colls” [5]
op (DTT, needs porting)
ap space;
re to known limits [5]
ytes” / known max (note: includes localhost traffic)
treat, say, 5+ as high utilization
m iostat/netstat/...



device (disk) utilization. For controllers, percent busy has much less meaning, so we’re calculating utilization based on throughput (bytes/sec) and IOPS instead. Controllers typically have limits for these based on their busses and processing capacity. If you don’t know them, you can determine them experimentally.

PMC == Performance Monitoring Counters, aka CPU Performance Counters (CPC), Performance Instrumentation Counters (PICs), and more. These are processor hardware counters that are read via programmable registers on each CPU. The availability of these counters is dependent on the processor type. See pmc(3) and pmcstat(8).

pmcstat(8): the FreeBSD tool for instrumenting PMCs. You might need to run a kldload hwpmc first before use. To figure out which PMCs you need to use and how, it usually takes some serious time (days) with the processor vendor manuals; eg, the Intel 64 and IA-32 Architectures Software Developer’s Manual, Volume 3B, Appendix A-E: Performance-Monitoring Events.

DTT == DTraceToolkit (http://www.brendangregg.com/dtrace.html#DTraceToolkit) scripts. These are in the FreeBSD source (http://svnweb.freebsd.org/base/head/cddl/contrib/dtracetoolkit/) under cddl/contrib/dtracetoolkit, and dtruss is under /usr/sbin. As features are added to DTrace see the freebsd-dtrace mailing list (https://lists.freebsd.org/mailman/listinfo/freebsd-dtrace), more scripts can be ported.

CPI == Cycles Per Instruction (others use IPC == Instructions Per Cycle).

I/O interconnect: this includes the CPU to I/O controller busses, the I/O controller(s), and device busses (eg, PCIe).

SOFTWARE RESOURCES

COMPONENT	TYPE	METRIC
Kernel mutex	utilization	lockstat -H (held time); DTrace lockstat provider
Kernel mutex	saturation	lockstat -C (contention); DTrace lockstat provider [6]; spinning shows u
Kernel mutex	errors	lockstat -E (errors); DTrace and fbt provider for return probes and erro
User mutex	utilization	DTrace pid provider for hold times; eg, pthread_mutex_*lock() return t
User mutex	saturation	DTrace pid provider for contention; eg, pthread_mutex_*lock() entry to
User mutex	errors	DTrace pid provider for EINVAL, EDEADLK, ... see pthread_mutex_lock
Process capacity	utilization	current/max using: ps -a l wc -l / sysctl kern.maxproc; top, "Processes:
Process capacity	saturation	not sure this makes sense
Process capacity	errors	"can't fork()" messages
File descriptors	utilization	system-wide: pstat -T, "files"; sysctl kern.openfiles / sysctl kern.maxfile
File descriptors	saturation	I don't think this one makes sense, as if it can't allocate or expand the
File descriptors	errors	truss, dtruss, or custom DTrace to look for errno == EMFILE on syscalls

lockstat: you may need to run kldload ksyms before lockstat will work (otherwise: "lockstat: can't load kernel symbols: No such file or directory").

[6] e.g., showing adaptive lock block time totals (in nanoseconds) by calling function name: dtrace -n 'lockstat:::adaptive-block { @[caller] = sum(arg1); } END { printa("%40a%@16d ns\n", @); }'

IN PRACTICE

You may notice that many metrics are difficult to observe, especially those involving pmcstat(8), which could take days to figure out. In practice, you may only have time to perform a subset of the USE method, studying those metrics that can be determined quickly, and acknowledging that some remain unknown due to pressures of time. Knowing what you don't know can still be enormously valuable: there may come a time when a performance issue is of vital importance to your company to resolve, and you are already armed with a to-do list of extra work than can be performed to more thoroughly check resource performance.

OTHER TOOLS

I didn't include procstat, sockstat, gstat or others, as the USE method is designed to begin with questions, and only uses the tools that answer them. This is very different from making a list of all the tools available and then trying to find a way to use them all. Those other tools are useful for other methodologies, which can be used after this one.

It's hoped that more tools are developed to make the USE method easier, expanding the subset of metrics that you have time to regularly check. If you are a FreeBSD developer, then you may be the first to develop a bustop, for example, which could be a PMC-based tool for showing busses and interconnects, and their utilization, saturation, and errors.

CONTINUED ANALYSIS

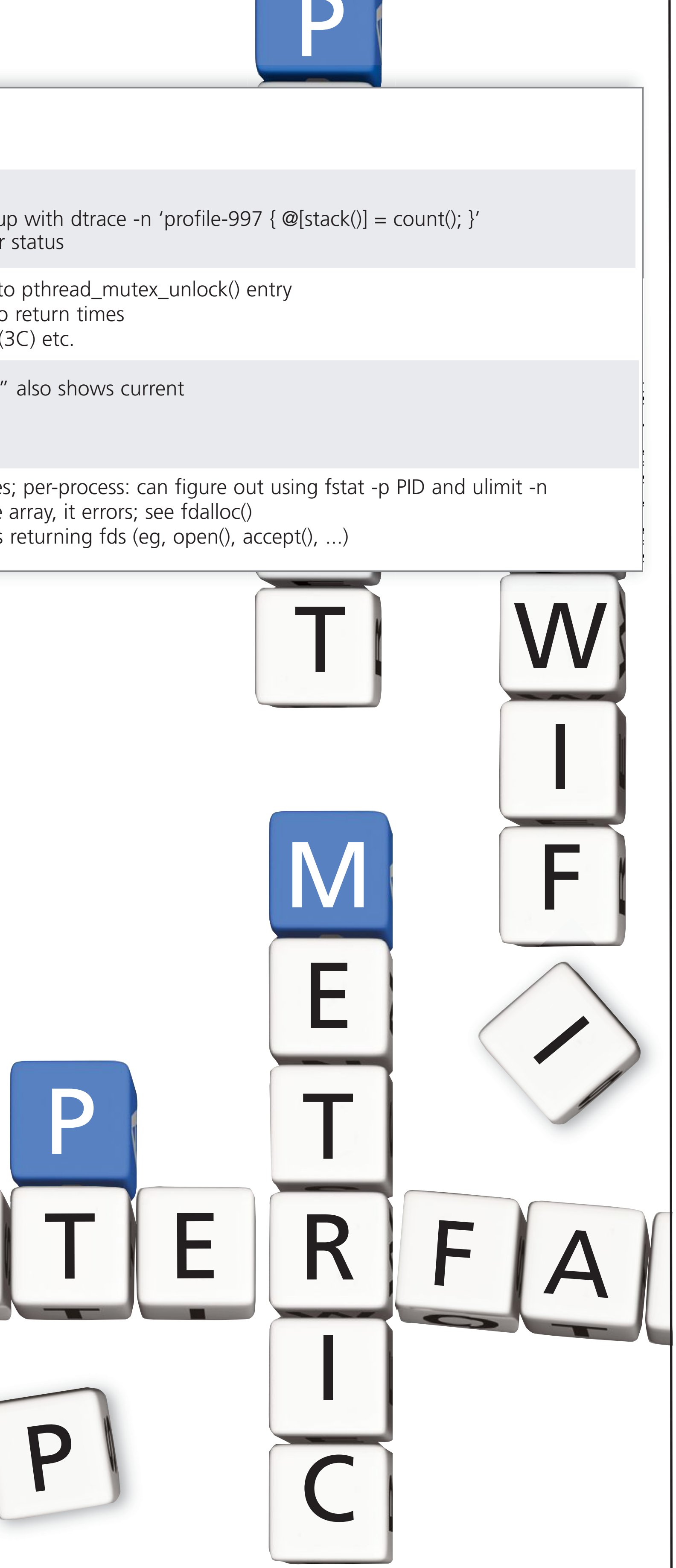
The USE method is intended as an early methodology to identify common bottlenecks and errors. These may then be studied using other methodologies, such as drill-down analysis and latency analysis. There are also performance issues which the USE method will miss entirely, and it should be treated as only one methodology in your toolbox.

REFERENCES

Thinking Methodically about Performance, Brendan Gregg, ACM Queue, vol. 10, no. 12, 2012

- The USE Method: FreeBSD Performance Checklist summary on my blog
- FreeBSD source code and man pages
- FreeBSD Wiki PmcTools
- FreeBSD Handbook

Brendan Gregg is a senior performance architect at Netflix. He is the author of the book *Systems Performance* (Prentice Hall, 2014), primary author of *DTrace: Dynamic Tracing in Oracle Solaris, Mac OS X,* and *FreeBSD* (Prentice Hall, 2011), and recipient of the USENIX 2013 LISA Award for Outstanding Achievement in System Administration. He was previously a performance lead and kernel engineer at Sun Microsystems where he developed the ZFS L2ARC and various DTrace providers.



up with dtrace -n 'profile-997 { @[stack()] = count(); }'
r status

to pthread_mutex_unlock() entry
o return times
(3C) etc.

" also shows current

es; per-process: can figure out using fstat -p PID and ulimit -n
e array, it errors; see fdalloc()
s returning fds (eg, open(), accept(), ...)

FreeBSD™ JOURNAL

NOW AVAILABLE AS A DYNAMIC EDITION!

The Dynamic Edition format offers subscribers the same features as the App, but permits viewing the Journal through your favorite browser.

Read It Today!

The DE, like the App, is an individual product. You will get an email notification each time an issue is released.

A one-year subscription is $19.99, and a single copy is $6.99—the same pricing as the App version.

www.freebsdfoundation.com

PORTSreport

by Frederic Culot

There has been quite a lot of activity on the ports tree during the past two months! And there's a lot of good news, too, with fresh committers joining the project and exciting new tools to ease our work on ports.

NEW PORTS COMMITTERS

Some of us committers remember the day when we received our commit bit as a very special day. I felt powerful at first because I could log in to the official FreeBSD hosts, but it quickly turned into anxiety, since—as is said—"with great power comes great responsibility!" The day has arrived for two new talents to join, Bartek Rutkowski and Stephen Hurd. I hope they will enjoy their work on FreeBSD as much as I do!

Another piece of good news is that Stefan Esser requested his commit bit be reinstated following a busy period during which he could not participate in the project. We were all pleased to have him back in our ranks. For those of you who may not know, commit bits are taken into safekeeping after an inactivity period of 18 months. When the idle committer is ready to come back, his commit bit is reinstated and he is usually assigned a mentor in charge of doing the ramp up and explaining the latest changes that have been applied to the ports tree.

CHANGES IN THE FreeBSD PORTS MANAGEMENT TEAM

There have been some significant changes in the ports management team as well—with some good and, unfortunately, some sad news too.

• ***Let's start with the good news:*** Steve Wills was granted full membership to the portmgr team. Steve has done tremendous work on ports, especially in the field of testing. Port managers believed providing Steve with full voting rights was a fair reward, and to enroll him in the team will also help secure future improvements in the field of ports testing.

• ***Now for the sad news:*** Our beloved Canadian Thomas Abthorpe decided to step down from his portmgr-secretary position. While I suspect this is secretly related to his pool of Canadian jokes hav-

NEW *TOOLS for tracking BUGS, etc.*

The past couple months brought us some exciting new tools for tracking bugs and easing the process of reviewing changes. Regarding bug tracking, the first discussions on the need to depart from GNATS actually began about 10 years ago! But the switch was finally made and we now rely on Bugzilla *(https://bugs.freebsd.org/bugzilla/)*. The transition still has a few corner cases that need some polishing, but so far the developers are quite happy with this switch to a more modern bug tracker. For those of you who are not yet accustomed to the way problem reports are managed at FreeBSD, I would advise you to start with *http://www.freebsd.org/support/bugreports.html* which provides useful links.

Now for code reviews, ports committers are currently experimenting with Phabricator *(https://phabric.freebsd.org/)* to ease the process of reviewing code. This is quite a change for FreeBSD, as no dedicated tools have been used until now, but bear in mind that no policy yet exists that requires reviews to be made via Phabricator. We are still experimenting with this new process, and if you are interested in experimenting with us, I would suggest you start with the explanations found on this page *https://wiki.freebsd.org/CodeReview.*

ing dried up, the official reason is that Thomas wants to focus more on his private and professional lives. Needless to say, the whole ports community is in mourning.

And to make things even worse, I was tricked into replacing Thomas and became the new portmgr secretary. Tabthorpe always says that one Canadian is at least worth four Frenchies, so we fall far short of the mark here, but nonetheless I will do my best to serve the ports community. ●

20th ANNIVERSARY

To complete this column, I want to mention an important event: the 20th anniversary of the FreeBSD ports tree! Indeed, it all started on August 21, 1994, with the following commit from Jordan Hubbard:

Commit my new ports make macros. Still not 100% complete yet by any means but fairly usable at this stage.

Indeed, those sets of macros seem to still be fairly usable 20 years late. Since the beginning, more than 500 committers have worked on the project, generating more than 300,000 commits to keep almost 25,000 ports up-to-date. A big thanks to all who have given some of their free time and energy to make this happen, and I am eager to see where we will be 20 years from now. In the meantime, for surprises on this anniversary day check our blog (http://blogs.freebsdish.org/portmgr/) and social networking pages!

http://fb.me/portmgr
https://twitter.com/freebsd_portmgr
https://plus.google.com/u/0/communities/108335846196454338383

Having completed his PhD in Computer Modeling applied to Physics, Frederic Culot has worked in the IT industry for the past 10 years. During his spare time he studies business and management and just completed an MBA. Frederic joined FreeBSD in 2010 as a ports committer, and since then has made around 2,000 commits, mentored six new committers, and now assumes the responsibilities of portmgr-secretary.

svn UPDATE

by Glen Barber

THE FREEBSD 9.3-RELEASE CYCLE

is nearing the final stages. FreeBSD 9.3 builds on the reliability and stability of FreeBSD 9.2 and brings a number of fixes and improvements to the 9.X series.

A list of changes since FreeBSD 9.2-RELEASE can be found here: http://www.FreeBSD.org/releases/9.3R/relnotes.html

ZFS Support Added to bsdinstall(8) stable/9@r264437

(http://svnweb.freebsd.org/changeset/base/264437)

Automatic installation onto a ZFS dataset was added to the FreeBSD installer, bsdinstall(8), near the end of the 10.0-RELEASE cycle. Revision 264437 of the stable/9 branch brings this and several other changes to bsdinstall(8), such as installing to a GELI-encrypted geom(4) provider and automatically aligning sectors to 4096 bits.

Loadable Xen Kernel Module stable/9@r266269

(http://svnweb.freebsd.org/changeset/base/266269)

Prior to FreeBSD 10.0-RELEASE, running FreeBSD as a virtual machine guest on the Xen hypervisor required specific changes to the kernel configuration. FreeBSD 10.0 natively supports the Xen environment, without requring special kernel configuration options. FreeBSD 9.3 will also natively support running as a Xen virtual machine by shipping with a kernel module, xenhvm.ko.

ttys(5) "onifconsole" Addition stable/9@r267243

(http://svnweb.freebsd.org/changeset/base/267243)

A new flag has been added to the ttys(5) file–"onifconsole." The new flag is equivalent to "on" (activating the serial console) when the tty is an active kernel console; otherwise, it defaults to "off."

This is particularly useful for embedded systems, where there may be one or more serial channels available, or on systems with IPMI SoL (serial over LAN) connections and the "default" tty may differ.

This change first appeared in head/ in revision 267243.

Process Filtering by Jail Name stable/10@r266280

(http://svnweb.freebsd.org/changeset/base/266280)

The top(1) utility has been updated to include a new flag, '-J', to filter the process table by jail(8) name or number. Prior to this change, all processes on the running system would be displayed; whether or not the process was run within a jail. With this change, it is now possible to limit the process list to show only the processes running within a specified jail. To limit the process list to display only processes running outside of a jail environment, specify the jail number '0'.

vt(4) Now Included in the GENERIC Kernel head/@r268045

(http://svnweb.freebsd.org/changeset/base/268045)

The vt(4) driver is now included in the GENERIC kernel configuration. The vt(4) driver is the system console driver that integrates with KMS (Kernel Mode Setting) graphics cards. Prior to the vt(4) driver it would not be possible to switch back to the system virtual terminals after the Xorg environment starts. This change adds a new loader(8) tunable, kern.vty, which activates the vt(4) virtual terminal driver when set to 'vt.'

As a hobbyist, Glen Barber became heavily involved with the FreeBSD project around 2007. Since then, he has been involved with various functions, and his latest roles have allowed him to focus on systems administration and release engineering in the Project. Glen lives in Pennsylvania, USA.

Testers, Systems Administrators, Authors, Advocates, *and of course* **Programmers** *to join any of our diverse teams.*

COME JOIN THE PROJECT THAT MAKES THE INTERNET GO!

★ DOWNLOAD OUR SOFTWARE ★

http://www.freebsd.org/where.html

★ JOIN OUR MAILING LISTS ★

http://www.freebsd.org/community/mailinglists.html?

★ ATTEND A CONFERENCE ★

• *http://www.bsdcan.org/2014/* • *http://2014.eurobsdcon.org/*

The FreeBSD Project

The FreeBSD FOUNDATION

conference REPORT

by Michael Dexter

BSDCan 2014 was an amazing experience as always, but one theme characterized this year's event more than any other: COORDINATION.

Never in my dozen years in the community have I seen such active dialog among the various BSD projects. From praise to constructive criticism, developers from all the projects engaged with one other in sessions and in the priceless BSDCan hallway track. Beginning with a project that is close to my heart, Peter Grehan announced at the FreeBSD DevSummit that the bhyve hypervisor would soon support NetBSD, rounding out its support for OpenBSD, NetBSD and Linux virtual machines. I can think of no better way for developers to see firsthand how each operating system works and to cross-validate code. Kudos to Peter, Neel Natu, John Baldwin and everyone else who has helped bhyve become such a useful feature in FreeBSD.

Continuing in the spirit of coordination, Abhishek Gupta of Microsoft's Hyper-V group was on hand to discuss with developers how to guarantee that FreeBSD is a first-class Hyper-V guest OS. From the sound of it, Microsoft appears to have more developers focusing on FreeBSD than Intel! Together, bhyve and Hyper-V represent compelling OS-native hypervisors, and rest assured, Windows virtual machine support in bhyve is under active development.

Matt Ahrens of the OpenZFS project gave his annual update on new ZFS features that are making their way into FreeBSD in order to keep FreeBSD a first-class ZFS platform. Of these features, ZFS "bookmarks" will enable ZFS replication without relying on snapshots as a unit of history. Just how quickly the OpenZFS project transitioned from post-Sun Microsystems confusion to solid, OS-agnostic contributions is remarkable. We all owe Matt, who has just received his FreeBSD commit bit our gratitude for his active participation in the BSD community at events like BSDCan and AsiaBSDCon.

Other DevSummit highlights included a clarification of FreeBSD's "long-term support" policy with the comforting recognition that the project had in fact been more or less adhering to the proposed 5-year policy. A formal affirmation of such a policy is a valuable marketing tool for everyone from vendors to end users. A suggestion was raised for separating the FreeBSD base into packages to allow for modular updating and deployment. Done right, this could be of great value to embedded FreeBSD efforts.

Two notable highlights of the FreeBSD Doc Sprints were the participation of Ingo Schwarze of the mandoc project, who committed FreeBSD's Igor documentation proofing tool to OpenBSD ports, and Allan Jude's formal entrance into the FreeBSD project with a documentation commit bit. Allan and Kris Moore have done a great job raising awareness of FreeBSD and other BSD projects with the BSDNow podcast and are demonstrating just how seamless community and code participation can be.

Though many of us were already exhausted from all-day discussions and late-night coding, it was finally time for the conference proper to begin. This saw an infusion of yet more wonderful people and continued engagement and coding. Security was a key topic with the FreeBSD Address-space Layout Randomization (ASLR), Capsicum and LibreSSL talks standing out as must-see events. Each talk was highly cross-pollinated by developers from different BSD projects with almost a sense of obligation to the Internet community as a whole, given BSD's key role in the development of the Internet.

The Embedded track comprised of ARM, MIPS64 and NAND flash storage talks and was also very timely given the changing nature of computing. Warner Losh went into great detail about how NAND flash storage works and how a broad range of reliability is available from the various flash technologies. This track even extended to a lunch time MIPS router hacking BOF lead by Sean Bruno. It is great that we have real Unix on really affordable hardware.

The closing auction was fun as always and the clouds broke on Sunday, allowing quite a few attendees to walk around Ottawa and Parliament before heading home. Some brave systems administrators opted to take the first BSD Certification Group BSD Professional exam and the feedback I heard was very positive. The BSD Professional exam is a hands-on exam designed to complement the BSD Associate exam that the BSDCG has offered for several years. This is an exciting development and is testament to the continued growth of the BSD community. ●

Michael has used BSD Unix systems since 1991 and has participated in the jail, sysjail, mult, Xen and most recently bhyve virtualization projects. He is an independent BSD author and support provider.

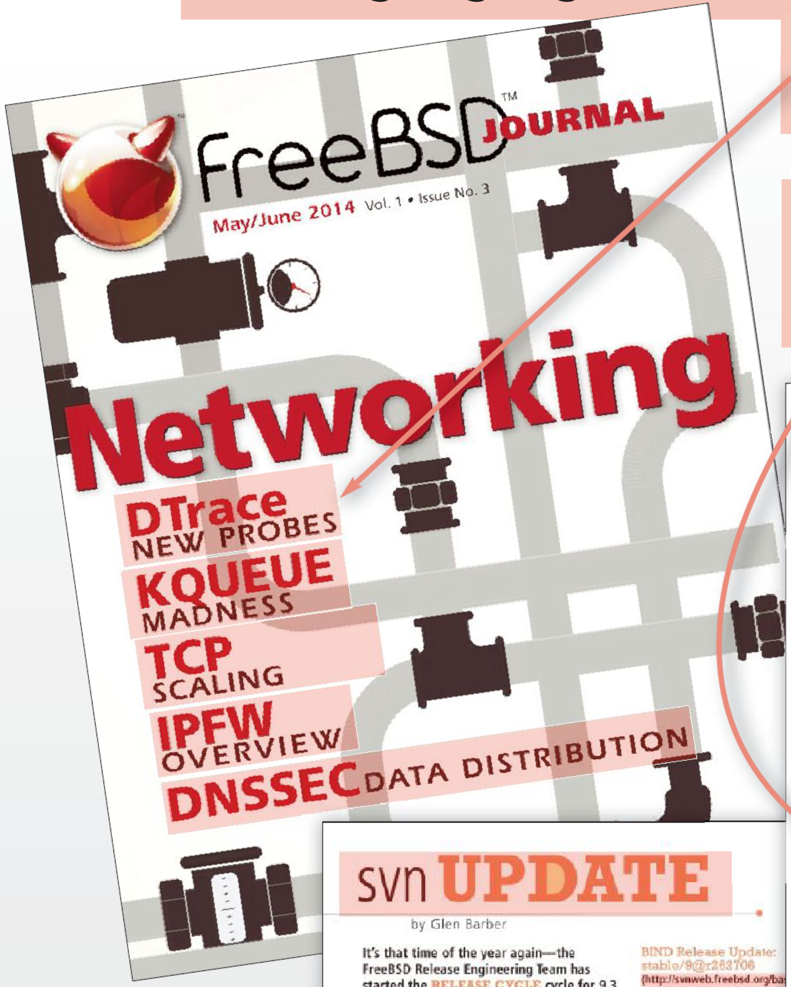
APP FEATURES

YOUR QUESTIONS ANSWERED

Q: "What's with the Blinking Red Highlights?"

A: **Blinking Highlights Indicate a Clickable Link.**

Example 1. **On the cover, click on any highlighted title to go to that article.**



Example 2. **Want to read an article or column listed in the table of contents? Just click on it.**

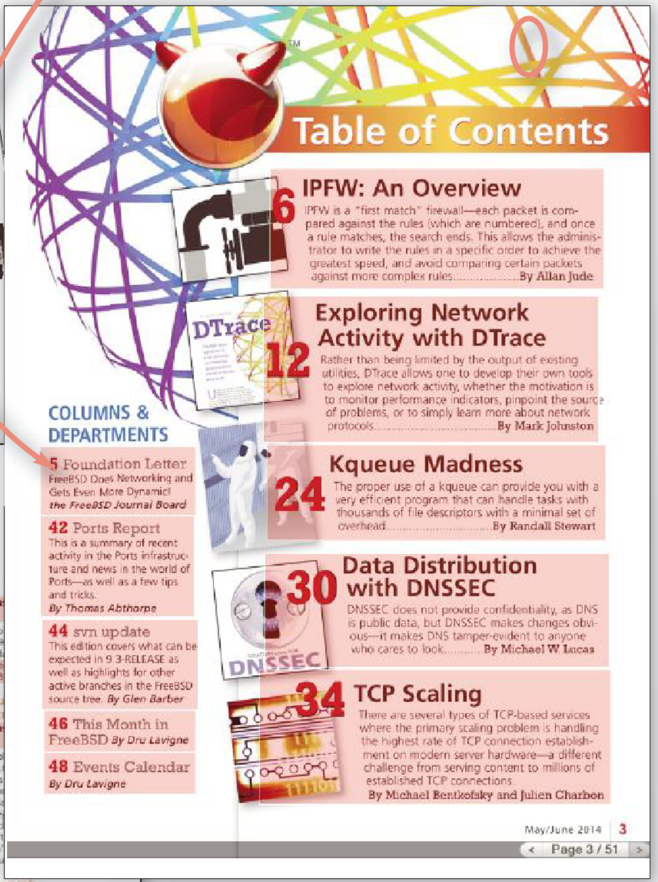


Table of Contents

May/June 2014 3

< Page 3 / 51 >

Example 3. **Click on any highlighted URL or word to connect to that website or access additional information.**

svn UPDATE

by Glen Barber

It's that time of the year again—the FreeBSD Release Engineering Team has started the RELEASE CYCLE cycle for 9.3 RELEASE. This edition of svn update covers what can be expected in 9.3-RELEASE as well as highlights for other active branches in the FreeBSD source tree.

ZFS Boot Support for Bhyve: head@r262331
(http://svnweb.freebsd.org/base?view=revision&revision=r262331)

The bhyve(8) hypervisor now has support for booting the virtual machine from a ZFS dataset allowing pure "root on ZFS" virtual machines. Prior to this change, it would have only been possible to boot from a UFS file system.

urndis(4) Support: stable/9@r262362
(http://svnweb.freebsd.org/base?view=revision&revision=r262362)

The urndis(4) driver has been added to stable/9. This driver was originally available in head/ with revisions r261541, r261543, and r261544, and provides Ethernet access over Remote NDIS, allowing mobile devices such as phones and tablets to provide network access via USB tethering.

The urndis(4) driver is also available in stable/10 as of revision r262363. The urndis(4) driver should support any USB RNDIS provider, such as those found on Android devices. The urndis(4) driver was ported from OpenBSD.

ext4 File System Support: stable/9@r262564
(http://svnweb.freebsd.org/base?view=revision&revision=r262564)

The ext2fs(5) code has been updated to enable ext4 file system support in read-only mode. Support for mounting read-only ext4 file systems was added in revision r262346 in head/.

Support for read-only ext4 file system is also available in stable/10 as of revision r262563.

BIND Release Update: stable/9@r262706
(http://svnweb.freebsd.org/ba
revision&revision=r262706)

The BIND DNS server in the stab
tem has been updated to version 9
release notes for this software upd
found here: https://lists.isc.org/pipe
bind-announce/2014-January/0008

pkg(8)Boostrap Signatu
Verification: stable/9@
(http://svnweb.freebsd.org/ba
revision&revision=r263038)

Signature verification code for p
strapping originally available in 10.
been merged to stable/9. With this
the administrator issues the 'pkg b
mand, the downloaded pkg(8) bina
will be verified against availabe ke
included in the base system, provid
security benefit for administrators t
wish to build the 'ports-mgmt/pkg'
the pkg(8) package management u

Radeon KMS Driver Support: stable/9@r263170
(http://svnweb.freebsd.org/base?view=revision&revision=r263170)

The Radeon KMS driver has been merged from head/. This supports Kernel Mode Setting (KMS) similar to the Intel KMS driver found on newer motherboards. This driver takes advantage of features available in newer Xorg drivers, such as xf86-video-ati.

VT Driver Merged: stable/9@r263817
(http://svnweb.freebsd.org/base?view=revision&revision=r263817)

The new system console driver, VT (also known as "NewCons"), has been merged from head/ to stable/9. The VT driver serves as a replacement for the sc(4) console driver, providing a number of enhancements ranging from UTF-8 font support, to enabling users running X11 with Kernel Mode

44 FreeBSD Journal

< Page 44 / 51 >

PLUS, Every AD PAGE hyperlink brings you to the advertiser's website.

this month In FreeBSD

BY DRU LAVIGNE

In this issue's Ports Report column, Frederic Culot notes that the FreeBSD Ports Collection will turn 20 on August 21, 2014. To commemorate this anniversary, I recently interviewed JORDAN HUBBARD, its creator.

DL *What prompted you to create the Ports System and what were your design goals? For example, what else, if anything, was available at that time for managing software on BSD or other UNIX variants and what features did your design want to address?*

JH I created the Ports Collection largely as a simple experiment in automation. At the time, I couldn't help but notice that all the software I tended to add to any new FreeBSD installation followed the same basic steps: Go grab the source tarball from some known location, extract it into a working directory, configure it, build it, and install the results.

This process was also generally considered "easy enough" that no one had really tried to automate the process, at least not in the sense of simply impedance-matching to whatever source location(s) and configuration/build processes the software used "natively" (vs. entirely repackaging it into some closed ecosystem), and this made me curious: Why were we being forced to remember all those various URLs for the source tarballs and manually do the same fetch/extract/configure/build steps—often many times—for each new FreeBSD install? It wasn't particularly hard, but it also seemed like one of those repetitive tasks for which computers were invented!

So anyway, that was really about it as far as the motivation was concerned. I just wanted to see if the process could be automated, and I used make(1) because it was already essentially designed to do that exact type of job-automate repetitive task. I just needed to write the make macros to allow one to create a port's makefile relatively succinctly.

I also didn't really approach the experiment with any "design process" in mind—I simply kept porting things, going after some of the more unusual/difficult to port pieces of software as a test of the overall concept, and evolved the make macros to meet the needs of the various challenges I ran into along the way. Once I got to about 200 ports or so, I concluded that the concept had been adequately proven and my curiosity entirely satisfied, so I approached Satoshi Asami, who was already contributing a lot of ports and seemed far more motivated than I to carry the ball forward. He's really the one who substantially bootstrapped the Ports Collection into what it is today. They didn't call him the first "portsmeister" for nothing!

DL *What was your long-term vision for ports? For example, did you expect this to become the de facto way of installing software on BSD, or were you expecting this to be a stage one in a longer-term plan?*

JH I'm not sure I can say that I had any long-term vision specifically for the Ports Collection, but I certainly had some longer-term goals for the notion of "software husbandry"—as I like to refer to it—as a whole. The Ports Collection and the corresponding package management tools (I also wrote the first version of those) were largely just components of those goals.

What I really wanted was for third-party software to be really easy to discover and install by end-users, with the "messy details" of actually getting it onto their systems (or off again) being abstracted away to the point where all they needed to do was fire up some package browsing tool, click on a suitable category (or search in a search box), and select the things they wanted from a menu in order to install (or deinstall) it. I also wanted the framework used to build that third-party software to be fully exposed and accessible to developers, since someone obviously needs to feed the other end of the pipeline in order to have a third-party software collection at all. I knew the collection would not grow unless it was relatively easy to add to and update. Some of those goals were fulfilled, others not, and still others only with caveats, and that's a good segue to your next question!

DL ***Knowing what you know now, is there anything you wish you had included in the design of the Ports System?***

JH Oh yes, definitely! I could probably list hundreds of things I wish I'd done differently, but for the sake of brevity, I'll just list what I consider to be the most significant ones:

1. I wish I had not used make(1). I knew the Berkeley make macro system backwards and forwards so it was very easy for me to use it at the time, but what I failed to consider was the fact that makefiles are also very difficult to manipulate programmatically, so you can't easily sweep through the entire collection and make wholesale changes when you want to rearrange/refactor things or do any kind of real analysis of the Port Collection as a whole. It also made "automation of the automation" (like "easy ports creator" tools and such) a lot harder, since makefiles don't lend themselves to introspection. As a data description format, they leave a lot to be desired.

2. The use of make(1) also broke a cardinal rule that I wasn't really aware of at the time (hey, it was the '90s—we were still idealistic). It mixed the active and the passive data together in one place, or if you prefer an English language metaphor, the "verbs" and the "nouns" describing a port and its actions were hopelessly intertwined in the same metadata. This made it effectively impossible to audit what a port was doing—or what it wished to do in advance—at build time. Since a lot of ports also need to execute at least their installation phases as root, that was a really unfortunate design choice from a security perspective. A dedicated port building machine can always do its work in a chroot or jail sandbox, but most users simply execute "make install" as root on their machines directly, and even if a port is not malicious in nature, it can still suffer from flaws in its construction that lead to unintended consequences.

3. I mixed the process of installing software with the process of building software. At the time, it seemed like the last logical step in the build process was to support the actual installation of the software (otherwise, what would be the point?), but that didn't really properly distinguish between the responsibilities of a build framework and a package-management framework. I later did some hacky impedance matching between the package tools and the Ports Collection such that either "make install" or "make package; pkg_add <resultingpackage>" would yield the same result, and leave behind the same registration information, but what I should have done in the first place was just make the output of the Ports Collection always be packages and never actually touch the host system, leaving that task entirely to the package management system.

DL ***Looking to the future, what are your thoughts on software management? Do you see ports as part of that future?***

JH Well, if I had to do it all over again today, I would probably describe a port in some machine-parseable format so that large-scale modifications of the tree could be done with less pain. I would make it possible for ports to be a bit more object-oriented (with inheritance and mixins). I would make the build machinery always execute in some kind of sandbox, with a variety of possible targets, e.g., not necessarily the same host architecture or release version of the builder. The end-result would always be a package, and I would also only allow packages to execute some finite set of actions at install (or uninstall) time, e.g., not just let them execute arbitrary, unauditable shell commands. That would mean all of the metadata for a package would be essentially "passive" until explicitly acted upon by the package management machinery, which could then also implement various security policies about which packages (or users) were allowed to do certain things.

I would also give a lot more thought to the actual runtime environment of the software being packaged. It's not the same Internet we had in the 1990s, obviously, and we can't just arbitrarily trust third-party software anymore—coding errors, bad actors, the whole security environment has changed. This means we need to establish provenance for everything we install on our machines, and even once we've established provenance, we need to still sandbox the heck out of it such that it can only access its own data, or that data to which we carefully grant it access, and not just run wild with our own permissions, or worse, on the system. All of those requirements are going to affect how software is audited, built, signed, distributed, and installed, and all the tools that FreeBSD is using need to evolve accordingly.

DL ***Any other points of interest or trivia regarding ports or software management?***

JH If software ecosystems like the Ports Collection have taught us anything, I think it's that some of our fundamental notions about software husbandry need a serious rethink. If you look at the Ports Collection today, it's a forest of individual software dependencies and versions, many of which are mutually incompatible, and even a well-organized junkyard is still a junkyard. It just keeps getting bigger and less wieldy as time goes on, and I'm not sure how much further it can scale out before it starts to collapse under its own weight. Some might argue that such collapse is already under way.

Moreover, by making it trivially easy to link things together in arbitrary ways, we've also only encouraged the perpetuation of some of the software industry's worst practices. Linking together lots of disparate software components into a single address space to create an application, whether it's targeted at some embedded role or running in a feature-rich environment like a desktop, might be easy and rather tempting to do, particularly when you're coding to a deadline, but it's also obviously fragile and fraught with potential peril.

The recent events with OpenSSL have amply demonstrated this, as if we needed more demonstration, and as significant as the fallout from the Heartbleed vulnerability has been, I think we've really only seen the tip of that iceberg. We're going to have even wider-spread vulnerabilities, and suffer even more pain before the industry stops thinking of dynamic libraries and loadable modules as "handy software ICs" and more as scary things for which they need to use opto-isolators everywhere. Software packages like OpenSSH and Postfix have been using multi-process, privilege-separated models for years now, but we need to think of ways to get everyone to do this as a matter of course, even if it requires the creation of fast and secure IPC mechanisms (and handier APIs that make those things easier to use) to facilitate it.

In that kind of world, it's just possible that software management frameworks like the Ports Collection could actually be part of the process of driving such best practices rather than the current worst practices. One can only hope! ●

Dru Lavigne is Director of the FreeBSD Foundation and Chair of the BSD Certification Group.

THE INTERNET NEEDS YOU

GET CERTIFIED AND GET IN THERE!

Go to the next level with BSD CERTIFICATION

Getting the most out of BSD operating systems requires a serious level of knowledge and expertise

NEED AN EDGE?

BSD Certification can make all the difference.

Today's Internet is complex. Companies need individuals with proven skills to work on some of the most advanced systems on the Net. With BSD Certification **YOU'LL HAVE WHAT IT TAKES!**

SHOW YOUR STUFF!

Your commitment and dedication to achieving the **BSD ASSOCIATE CERTIFICATION** can bring you to the attention of companies that need your skills.

BSDCERTIFICATION.ORG

Providing psychometrically valid, globally affordable exams in BSD Systems Administration

THROUGH OCTOBER

BY DRU LAVIGNE

2014 Events Calendar

These BSD-related conferences are scheduled for the third quarter of 2014. More information about these events, as well as local user group meetings, can be found at **bsdevents.org.**

Fossetcon • Sept. 11–13, 2014 Orlando, FL

http://www.fossetcon.org • Fossetcon is the Free and Open Source Software Expo and Technology Conference. There will be a FreeBSD booth in the expo area and the BSDA certification exam will be available.

EuroBSDCon • Sept. 25–28, 2014 Sofia, Bulgaria

http://2014eurobsdcon.org • EuroBSDCon is the premier European conference on BSD operating systems. It features two days of developer summits and tutorials followed by two days of technical presentations.

Grace Hopper Celebration • Oct. 8–10, 2014 Phoenix, AZ

http://gracehopper.com • The FreeBSD Foundation will have a booth in the expo area of the Grace Hopper Celebration of Women in Computing. FreeBSD will also participate in the Open Source Day and will provide an overview of how to get involved in the FreeBSD Project.

Ohio LinuxFest • Oct. 24–26, 2014 Columbus, OH

https://ohiolinux.org/ • There will be a FreeBSD booth and several FreeBSD-related talks at the 12th annual Ohio LinuxFest. The BSDA certification exam will also be available on Sunday, October 26.

Keep FreeBSD Free!
(Donate Today...)

Your donatoins have helped FreeBSD by:

Funding Projects to Advance FreeBSD
Development Projects include: Updated video console (Newcons), UEFI system boot support, Native iSCSI kernel stack, New automounter, Capsicum improvements.

Hiring Additional Staff Members
The Foundation now has additional support with marketing and promotion.

Educating the Public and Promoting FreeBSD
We continue to provide funding and leadership through publishing the FreeBSD Journal, creating high-quality educational brochures, sponsoring conferences, and visiting companies to help facilitate collaboration efforts with the Project.

Providing Legal Support to the Project
You're helping us protect FreeBSD IP and provide legal guidance for the Project.

Building FreeBSD Project Infrastructure
Your funds allow for hardware purchases to keep the project growing.

What will your donation help the Foundation accomplish this year?

Support FreeBSD by donating to the FreeBSD Foundation

FreeBSD is internationally recognized as an innovative leader in providing a high-performance, secure, and stable operating system. Our mission is to continue and increase our support and funding to keep FreeBSD at the forefront of operating system technology.

Thanks to people like you, the FreeBSD Foundation has been proudly supporting the FreeBSD Project and community for over 14 years now. We are incredibly grateful for all the support we receive from you and so many individuals and organizations that value FreeBSD.

Make a gift to support our work throughout the rest of 2014. Help us continue and increase our support of the FreeBSD Project and community worldwide!

Making a donation is quick and easy. Go to:
www.freebsdfoundation.org/donate

The FreeBSD FOUNDATION

www.freebsdfoundation.org